VIRTUALIZED DYNAMIC RESOURCE ALLOCATION ALGORITHM FOR THE INTERNET DIFFSERV DOMAINS

AHMED JUMAA LAFTA AL-WASITY

INFORMATICS RESEARCH CENTRE

SCHOOL OF COMPUTING, SCIENCE AND ENGINEERING

UNIVERSITY OF SALFORD

SALFORD, UK

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS OF THE DEGREE OF DOCTOR OF PHILOSOPHY, JUNE 2017

Abstract

The Differentiated Services (DiffServ) architecture has been proposed for providing different levels of service to the Internet Protocol (IP) traffic. Current discussions in the DiffServ networks are focused on managing resources dynamically according to the traffic conditions of the DiffServ router (Per Hop Behaviour). Software Defined Networks (SDN) and Network Function Virtualisation (NFV) technologies have recently emerged in the research agenda to support researchers in managing network domains and to achieve better use of domain resources. This thesis introduces a new scheduling algorithm called "Dynamic Resource Allocation Management - Network Function Virtualization (DRAM-NFV)" to allocate the service classes resources in the proportional delay DiffServ domains. DRAM-NFV algorithm manages the resources among service classes within the edge routers of the DiffServ domains dynamically according to their traffic conditions and manages these resources between the DiffServ domains in the event of congestion based on their traffic conditions at the egress routers of the upstream domain and ingress routers of the downstream domain. The NFV executes the DRAM-NFV algorithm on a virtualized - Network as a Service (NaaS) - cloud infrastructure to manage the SDN controllers for the edge routers of the DiffServ domains through monitoring the traffic conditions in the service classes at the edge routers and reallocating the out-link resources of the edge routers among service classes.

A number of test scenarios were conducted in this research in order to test the performance of the DRAM-NFV algorithm. The performance of DRAM-NFV algorithm is compared with the performance of the DWFQ algorithm by comparing the average End to End Delay for service classes traffic and links utilization. The DWFQ algorithm cannot manage resources between DiffServ domains but can manage the resources locally and dynamically for each DiffServ domain separately.

The network simulator NS3 has been used to implement these test scenarios and to test the performance of the DRAM-NFV algorithm. The results show that with the DRAM-NFV algorithm, better balance for DiffServ domains resources can be achieved through monitoring the bandwidth hungry service class at the downstream domain and managing its resources at the upstream domains. As a consequence of this, the utilizations of some service classes traffic are improved and the average End to End Delay for overall traffic are also reduced. An example of the improvement that was achieved by managing resources between (upstream and

downstream) DiffServ domains dynamically, in test scenario 3- Case Study 2, the average utilization for the highest priority class (SC_1) for whole period of simulation at the destination end is increased by 0.175% and the average End to End Delay for overall traffic is also reduced by 800 msec. As a result of reducing the average End to End Delay for overall traffic and improving the utilizations of service classes traffic, the QoS of applications traffic can be improved during the congestion periods in DiffServ domains.

Acknowledgements

The process of undertaking a doctoral study for four consecutive years can be considered as a journey full of difficult challenges and unique experiences. In October 2013, I remember I spent some months to discover the research topic that I was to focus on, define the research problem and suggest a primitive solution for it. This was especially fraught as the subject of my research is cutting edge and there were few other studies in the field at that time. I can also remember days of disappointment and discouragement and sleepless nights when I was unable to achieve progress in my research and tackle the difficulties that I was facing in the research. At the same time, I felt great satisfaction when some parts of the research were completed and when finding solutions to the difficulties.

I had the great fortune to be guided during this journey by Dr. Adil Al-Yasiri. I hereby express my deep sense of gratitude to him for encouraging me, for his commitment to hold regular meetings every month to follow up the progress in this research, his thoughts, and his advice and recommendations which made this doctoral thesis possible. Throughout his supervision, I learned from his opinions and suggestions on how to process problems in scientific research. His constructive criticisms have contributed immensely to the evolution of my ideas about this research. Also I have benefited from his ideas and advice on developing my academic research skills.

Sincere thanks and gratitude go to my mother, father and my elder brother Dr. Mohammed who all live in Baghdad. They have always encouraged me to travel outside Iraq to complete my studies and they have always provided me with invaluable support. My special appreciation goes to my mother. Although her illness makes her unable to perform her activities normally, she is always praying for me and asking God to facilitate my success and help me in my studies. I am very much thankful to my brother who has taken care of the health of our parents and has taken on many other tasks for the whole duration of my study. I know well, that my family will be happy when I get the doctoral degree.

I would like to express my deep thanks to the Office of the Prime Minister of the Republic of Iraq - the Higher Committee for Educational Development in Iraq (HCED-IRAQ)[1], the official sponsor of this scholarship for the obtaining of a doctoral degree in the field of Data Telecommunication and Networks from the University of Salford – Manchester, United Kingdom. Also, I would like to thank so much the Cultural Attaché of the Iraqi Embassy in London, UK and Al-Nahrain University – Baghdad, Iraq for their supports during the Ph.D. study period.

Special thanks go to all my relatives in the United Kingdom.

I am also grateful to many people that offered significant help and advice over the last four years.

Finally, I would also like to thank the members of my thesis committee and the Informatics Research Centre, School of Computing, Science and Engineering in the University of Salford for the use of their facilities.

With my kindest wishes….

Ahmed Jumaa Lafta Al-Wasity
June 2017
Greater Manchester – UK

1 On January 19, 2009, the Office of the Prime Minister of the Republic of Iraq - the Higher Committee for the Education Development in Iraq (HCED-IRAQ) launched an educational initiative to enhance the educational system of Iraq. Many scholarships were given for thousands of qualified graduate Iraqi students to study in the United Kingdom, the United State, and Australia in order to obtain masters and doctorate degrees in various scientific and literary disciplines.

Table of Contents

List of Figures

List of Tables

Abbreviations

A-CPI	Application-Controller Plane Interface
AF	Assured Forwarding
APIs	Application Program Interfaces
BE	Best Effort
BPR	Backlog Proportional Rate
BW	Bandwidth
C-DPI	Controller-Data Plane Interfaces
CAPEX	Capital Expenditure
CBQ	Class Based Queuing
CPU	Control Processing Unit
DBWS	Dynamic Benefit Weight Scheduling
DDP	Delay Differentiated Parameters
DiffServ	Differentiated Services
DRAM-NFV	Dynamic Resource Allocation Management -Network Function Virtualization
DSCP	Differentiated Service Code Point
DWFQ	Dynamic Weighted Fair Queuing
EF	Expedited Forwarding
ETSI	European Telecommunications Standards Institute.
FIFO	First Input First Out
FTP	File Transfer Protocol
GCD	Greatest Common Divisor
GPS	Generalized Process Sharing
H-PFQ	Hierarchical Packet Fair Queuing
HaaS	Hardware as a Service
HPD	Hybrid Proportional Delay
I-CPI	Intermediate-Controller Plane Interface
IaaS	Infrastructure as a Service
IETF	Internet Engineering Task Force
IP	Internet Protocol
MPLS	Multi-Protocol Label Switch
NaaS	Network as a Service
NFs	Network Functions
NFV	Network Function Virtualisation
NOS	Network Operating System
NS2	Network Simulator - version 2
NS3	Network Simulator – version 3
OPEX	Operating Expenses
PaaS	Platform as a Service
PAD	Proportional Average Delay
PDD	Proportional Delay Differentiation
PHB	Per-Hop Behaviour
PQCM	Proportional Queue Control Mechanism
QoS	Quality of Service
RAM	Random Access Memory

RED	Random Early Detection
RIO	Random Early Detection with In/Out bit
SaaS	Software as a Service
SC	Service Class
SDN	Software Defined Network
TCP	Transmission Control Protocol
UDP	Unit Datagram Protocol
VLAN	Virtual Local Area Network
VMs	Virtual Machines
VoIP	Voice over Internet Protocol
WFQ	Weighted Fair Queuing
WRED	Weighted Random Early Discarding
WTP	Waiting Time Priority

Chapter One
Introduction

1.1 Introduction:

Quality of Service (QoS) is the ability of a network to provide different types of services to various identified network traffics over different underlying technologies. Bandwidth, delay, delay variation (jitter) and packet loss rate are the parameters used to measure and specify QoS (Shaikh et al. 2002). The maximum bit rate that can be sustained between two end points of a network is defined as the bandwidth of the network link. Delay is the elapsed time incurred when a packet is passed from a sender through a network domain to its destination (end-to-end delay). Jitter is the variation in end-to-end, transient delays. Finally, the packet loss rate is defined as the ratio of dropped packets to the total number of packets (Kurose & Ross 2013).

The importance of Quality of Service (QoS) technologies has increased rapidly with the rapid development of multimedia applications over the Internet – such as distance learning, Video traffic, voice over Internet protocols, and so on. DiffServ is a QoS model, (Giordano et al. 2003). This model has been proposed and standardized to address the QoS requirements of real time applications (Blake et al. 1998). DiffServ provides differentiation services for aggregates of network traffic at routing decision points (Kusmierek et al. 2002).

Packet scheduling is one of the functions performed by a router and it is an important process carried out by DiffServ networks in order to allocate resources to service classes (Tanenbaum & Wetherall 2011). NFV and SDN are cutting-edge technologies in networking which co-operate in order to deploy network functions with less hardware dependence and as a software deployed on a virtualised infrastructure (Cloud) (Chiosi et al. 2012). The SDN and NFV technologies bring specific benefits with regards to efficient resource usage and faster operations management.

1.2 Differentiated Services:

Different types of QoS models, Differentiated Services (DiffServ) is one of them. It aims to achieves differentiation among service classes in the network through sequence of operations (Classification, Queuing or Dropping and Scheduling) based on the QoS requirements of the Internet applications and it is Per Hop Behaviour (PHB) such that each domain or router manages its resources locally depending on its traffic condition and in a different way from other domain.

Differentiated Services are beneficial in packet networks (Dovrolis 2000) because not all applications have the same QoS requirements. For example, viable Video traffic sessions necessitate lower delays and fewer losses than file transfers via FTP. With differentiation services, more demanding applications (or users) are provided with better performance than others.

The Internet Engineering Task Force (IETF) had the DiffServ architecture developed in the late nineties (Blake et al. 1998) (Nichols et al. 1998). The concept of DiffServ is that individual flows (microflows) with similar QoS requirements can be aggregated into larger traffic sets called (macroflows) or service classes. DiffServ uses a manageable, scalable and easily deployable architecture for differentiation services across IP networks. The network bandwidths in DiffServ are allocated to macroflows rather than to individual flows. All the packets in a macroflow experience the same forwarding behaviour in the routers. Each macroflow uses a specific service class queue or per-hop behaviour (PHB).

DiffServ can be provided by a set of routers forming an administrative domain called a DiffServ domain. The administrator defines a set of service classes each with corresponding forwarding rules. The classes may differ in terms of their requirements for delay, jitter, and the probability of packets being discarded in the event of congestion (Tanenbaum & Wetherall 2011). Different administrative DiffServ domains are interconnected to constitute the Internet (Reichmeyer & Networks 1998). Each DiffServ domain has core routers which are connected together in order to forward packets. The administrative DiffServ domains and end users are interconnected to each other through edge routers which lie at the boundary of the DiffServ domains (Kurose & Ross 2013). Figure 1-1 shows a typical DiffServ architecture as used in the Internet.

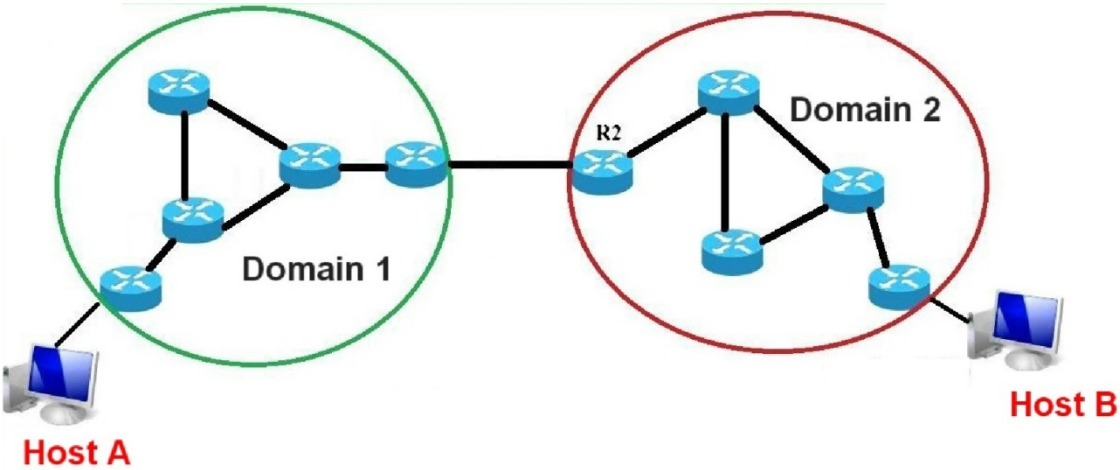


Figure 1-1, A typical DiffServ domains architecture.

When packets of a flow enter a DiffServ domain, they are assigned a code which specifies their forwarding priority (service class); this is achieved by means of a marker which is set by the edge router (Blake et al. 1998). This code is termed the packet's Differentiated Service Code Pint (DSCP). The aggregation of flows is achieved at the edges of a DiffServ network. A flow classification system (Suter et al. 1999) (Srinivasan et al. 1999) is required at the network edges in order to aggregate microflows into macroflows.

The aggregation of microflows into macroflows makes the DiffServ network model scalable and manageable, and with an easily deployable architecture because a state is required only for a few service classes not for each individual flow. In addition, there is a queue for each class instead of a queue for each flow. Consequently, the per queue operations of classification, scheduling, buffer management and policing become noticeably simpler and faster.

The DiffServ architecture does not require a signalling protocol because there are no resource reservations for individual flows. The DiffServ architecture allows both absolute (quantitative) and relative (qualitative) service differentiation.

The absolute service differentiation aims to provide a macroflow with a quantitative performance level (minimum forwarding rate, or maximum loss rate) at certain links or network paths. Minimum Forwarding rate means the minimum number of packets that can be sent at certain link or network in one second while the maximum loss rate means the maximum percentage of packets lost with respect to packets sent to certain links or network. Such an absolute differentiation requires some form of admission control to prevent users from sending traffic at a higher rate than their network contract allows.

On the other hand, the relative differentiation service provides a number of classes with preferential performance. The higher classes will provide better QoS to flows than the lower classes will (Dovrolis & Ramanathan 1999) – in terms of both queuing delays and packet losses. However, the exact QoS for each class is not specified and depends on the traffic conditions and the DiffServ mechanisms that the network deploys. The relative differentiation model does not require resources reservation, signalling, fixed routing, or admission control and so it is considered simpler to manage and to deploy. The relative differentiation model is used as the differentiation model in this research [as will be illustrated in chapter 2] because the absolute DiffServ domains aim to provide quantitative performance for their service classes thus this model does not allow us to modify the configuration parameters of absolute DiffServ domains that are setup by the domain administrator. While in relative DiffServ domains; it provides qualitative performance for their service classes and allow us to amend their configuration parameters. In addition, the resources are allocated depending on traffic conditions in service

class queues and the configuration parameters of a DiffServ domain thus this will help us on managing resources between relative DiffServ domains through monitoring the traffic condition in one domain and modifying the configuration parameters of another DiffServ domain.

1.3 Research Problem:

A packet in the Internet travels from source to destination by passing through one or more DiffServ domains. The problem that we have in the current approach of a DiffServ domain is that it is a static, it means that each router or domain allocates its resources locally or separately based on its traffic condition and the domain configuration parameters. Once the domain administrator sets the configuration parameters of a domain, these parameters will not change. The traffic conditions within a DiffServ domain can be changed continuously and randomly therefore there will be a situation that the DiffServ domain favoring certain amount of traffic while other network traffic will be suffered although the priority levels that we have made for service classes in a DiffServ domain so it needs to adjust the parameters at one domain to rebalance the resources at another domain.

In order to explain the research problem in more details, Figure 1-2 shows a packet which needs to be sent from host A in Domain 1 to host B in Domain 2; so the Domain 1 is considered the Upstream domain and Domain 2 is considered the Downstream domain based on traffic direction from Domain1 to Domain 2. Let us assume that this packet was marked with a high forwarding priority class when it entered Domain 1 (upstream) – it has been marked according to router strategy (traffic condition and configuration parameters) for Domain 1 or - per-hop behaviour. When the packet crosses over the boundary between two DiffServ domains, the resources at Domain 2 (downstream) are also allocated depending on the traffic condition at each edge DiffServ router in Domain 2 and configuration parameters for Domain 2; thus the forwarding probability of the packet may be altered if there aren't enough available resources for the service class to which the packet belongs at the next domain (Domain 2). That is, in DiffServ, all the service class queues of a router share one out link to forward their packets to the next hop (router or domain) and the traffic conditions within a DiffServ domain can be changed continuously and randomly so when the traffic of a certain service class or classes increases at Domain 1 (upstream), the traffic congestion could occur at the link that connects the upstream and downstream DiffServ domains and at the out links of DiffServ routers of Domain 2 for this class or these classes. This congestion causes the out links resources for Domain 2 and resources of the link that connects DiffServ domains to be mostly occupied by

the increasing or congested traffic and makes the out links' resources (queues buffer resources), allocated for other classes at the DiffServ routers in Domain 2 (downstream), not well utilized during the congestion period. So the research problem is to "How to manage resources between differentiated Services domains dynamically instead of managing resources statically for each domain in order to achieve better QoS for application traffic". Appendix A-1 presents an experiment to illustrate the research problem.

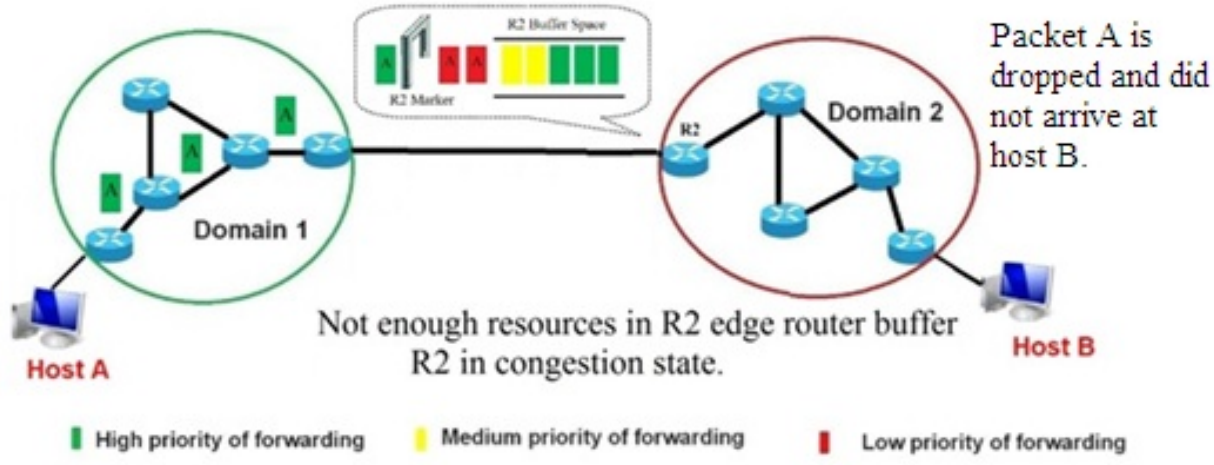


Figure 1-2, Research problem.

1.4 SDN and NFV Technologies:

Software Defined Networks (SDN) and Network Function Virtualization (NFV) are considered cutting-edge technologies. SDN is considered a valuable, emerging networking model that ensures that the control layer of a network device is separated from its data layer (Kate Greene 2009). In addition, SDN controllers can connect with one another and with the data layer of SDN forwarding device via Application program Interfaces (APIs) (Kreutz et al. 2015).

NFV has recently emerged and offers the implementation of network services as software executed on a virtualisation infrastructure (the Cloud). Consequently, it reduces hardware dependency (Chiosi et al. 2012). This method of implementation leads to efficient resource usage and faster operations management with lower costs because the NFV provides the infrastructure in which the application programs of the SDN controller are executed. Moreover, it (NFV) participates in the deployment of the controller's applications across multiple networks.

Although the DiffServ is Per Hop Behaviour (PHB) and it is a static approach in allocating its resources separately per each domain such that the domain administrator sets the parameters for service classes in a fixed manner and did not change it. It will be better if the DiffServ domains are managing their resources dynamically rather than statically because the network

operation (traffic condition) within a DiffServ domain can be changed continuously and randomly and therefore there will be a situation that the network favouring certain amount of network traffic. As a consequence, other network traffic will be suffered.

Instead of having a static DiffServ, we can monitor traffic condition in another domain and make changes on managing resources of other domain so we have now calibrating between DiffServ domains in managing resources. This calibration can be done by:

1. Using SDN switches, OpenFlow, for example that we can programmed them automatically and remotely. As a result of using SDN networks, we can monitor and manage the queues and congestion levels in DiffServ domains.

2. Using the concept of NFV in order to extract and implement virtually a certain function in DiffServ which is resource management function. As a result of using NFV, the management process will be deployed and do not become locally. It is still locally if the traffic condition within a DiffServ domain is normal but it becomes across DiffServ when there is unbalanced in traffic condition or congestion within a DiffServ domain.

1.5 Research Motivation:

This SDN and NFV techniques have not been used, as yet, for the implementation of the management of resources across DiffServ networks. Making use of the opportunities presented by NFV and SDN, the management process could take place outside the DiffServ domain itself. It could take place in a virtualized infrastructure while NFV monitors the traffic situation of the downstream domain (the domain that the packets will travel to) and provides the allocating resource scheme to the upstream domain.

The motivation for this current work is to make effective use of the benefits offered by NFV and SDN and to study how these techniques may be used to manage resources within the edge routers of DiffServ domains and among different DiffServ domains. Within the edge routers, the resources are allocated depending on the traffic state in each service class queue and according to the differentiation patterns within the DiffServ domain. Also, the management of resources among DiffServ domains includes the monitoring of the congestion level for each service class queue at the downstream domain. When the service class congestion level exceeds the congestion threshold parameters of a service class queue, the scheduling rate for the equivalent or mapped service class queue at the egress router of the upstream domain is reduced by an optimum value. In this present study, a new scheduling algorithm is proposed, termed "**Dynamic Resource Allocation Management – Network Function Virtualization**"

(**DRAM-NFV**). This algorithm has been designed to manage the resources within and across DiffServ domains. The DRAM-NFV algorithm is based on the Dynamic Weighted Fair Queuing DWFQ algorithm which, as currently implemented/used, is not able to manage the resources across different DiffServ domains. The recently established technologies of the Cloud and Software Defined Networks (SDNs) participate in the deployment of the resources management process 'virtually' across DiffServ domains and within the edge routers of DiffServ domains, using the concept of Network Function Virtualization (NFV).

1.6 Research Aims and Objectives:

The aim of this project is to make use of the NFV and SDN technologies in order to manage resource allocation within and among DiffServ domains so that better resource management for service class queues and greater fairness in the distribution of surplus bandwidth during the events of congestion within the DiffServ domains are achieved.

The key objectives of this research are as follows:

1. Explore a Dynamic Resource Allocation Management – Network Function Virtualization DRAM-NFV algorithm which is intended to manage the resources within and among DiffServ domains, making use of the NFV and SDN concepts.
2. Build a mathematical model that represents the DARM-NFV algorithm.
3. Implement a prototype of the DRAM-NFV algorithm and use a simulation environment to run a number of test scenarios for the validation and evaluation the DRAM-NFV algorithm.
4. Identify conditions which enable the DRAM-NFV algorithm to manage resources across DiffServ domains.

1.7 Research Questions:

Throughout the course of this research, the following questions have been addressed:

1. How can the resources across different DiffServ domains be managed?

2. What viable test scenarios can be proposed to test the performance of the DRAM-NFV algorithm?

1.8 Research Methodology:

The approach (as shown in Figure 1-3) adopted in this research can be itemised as follows:

1. Review previous research:

Previous relevant works have been reviewed in order to gain a good understanding of the research field and to investigate issues related to DiffServ networks, the Cloud, SDN and NFV. The various DiffServ models and the methods of allocating resources used by the proportional delay relative differentiated service model are presented in this thesis. This review of previous work also includes an identification of the most appropriate infrastructure for implementing the NFV which is the Network as a Service (NaaS) Cloud model. In addition, many areas of research that have emerged recently in the telecommunication and computer networks field which use the cutting-edge techniques of SDN and NFV are examined.

2. Identify the research problem:

The research proper starts by identifying the drawbacks associated with DiffServ networks. The current proportional DiffServ model allocates resources purely depending on the traffic conditions prevalent at each edge DiffServ router and so does not enable the management of resources across different DiffServ domains. Thus, when a packet crosses the boundary between two domains, the forwarding probability of the packet may be changed, inappropriately, because there are not enough available resources, at the next domain, for the service class that the packet belong to.

In identifying the research problem, it was necessary to consider the foundation on which a new resource allocation algorithm, called the DRAM-NFV algorithm, should be developed. Such an algorithm, clearly, must be capable of dealing with the issues identified and improving the performance of a proportional delay DiffServ by managing resources within a DiffServ router and also across different DiffServ domains in the event of congestion.

3. Define the parameters relevant to resource allocation management:

The parameters of the DRAM-NFV algorithm must be defined so that they construct a mathematical model for the algorithm. This mathematical model has to express the behavior of the proportional delay DiffServ model when using the DRAM-NFV algorithm and was used to implement a prototype to test and evaluate the performance of the DRAM-NFV algorithm within the simulation environment ns3.

4. Design a framework for the system.

To make the DRAM-NFV algorithm able to manage the resources within and across DiffServ domains, a framework is required which enables the DRAM-NFV algorithm to work. Both

Cloud and SDN technologies participate in supporting the DRAM-NFV algorithm. The Network-as-a-Service NaaS Cloud which allows for the virtualization of computing and networking resources in a Cloud environment is considered as an infrastructure for the DRAM-NFV algorithm. SDN is different from traditional networking models in that the SDN controller is not bundled with the SDN forwarding device and so can be represented as a software setup on the NaaS Cloud model. Deploying and communicating between instances of SDN controllers on the Cloud enables consolidated management of the DiffServ domain routers, increases network efficiency, reduces the cost of equipment, and reduces energy consumption.

5. Simulate the designed framework and run a number of test scenarios.

To achieve the research aim, test scenarios must be implemented and presented within a simulation environment in order to test the performance of the DRAM-NFV algorithm. The scenarios must take into consideration different topologies of DiffServ domains and enable the study of the performance of the DRAM-NFV algorithm when at least some of the DiffServ domains cannot manage their own resources appropriately via the NFV technology.

6. Evaluate and validate the DRAM-NFV algorithm.

The results of implementing the test scenarios via the network simulator, ns3, must be studied to evaluate the performance of the DRAM-NFV algorithm. The performance of the DRAM-NFV algorithm is to be compared with the performance of the DWFQ algorithm under the same simulation conditions. The DWFQ algorithm cannot manage resources across DiffServ domains – it allocates the resources dynamically and separately within each DiffServ router.

7. Make any modifications necessary to improve the DRAM-NFV algorithm.

If modifications are necessary to improve the performance of the DRAM-NFV algorithm, then this step allows for such modifications (to the algorithm and to its implementation) to take place according to the evaluation of the results which are produced by the experiments carried out in step 6.

8- Draw conclusions and recommendations and disseminate results.

Conclusions and recommendations must be drawn up after the results have been finalized. Here, the results are presented for the final time so that the thesis can be written up and the completed work presented.

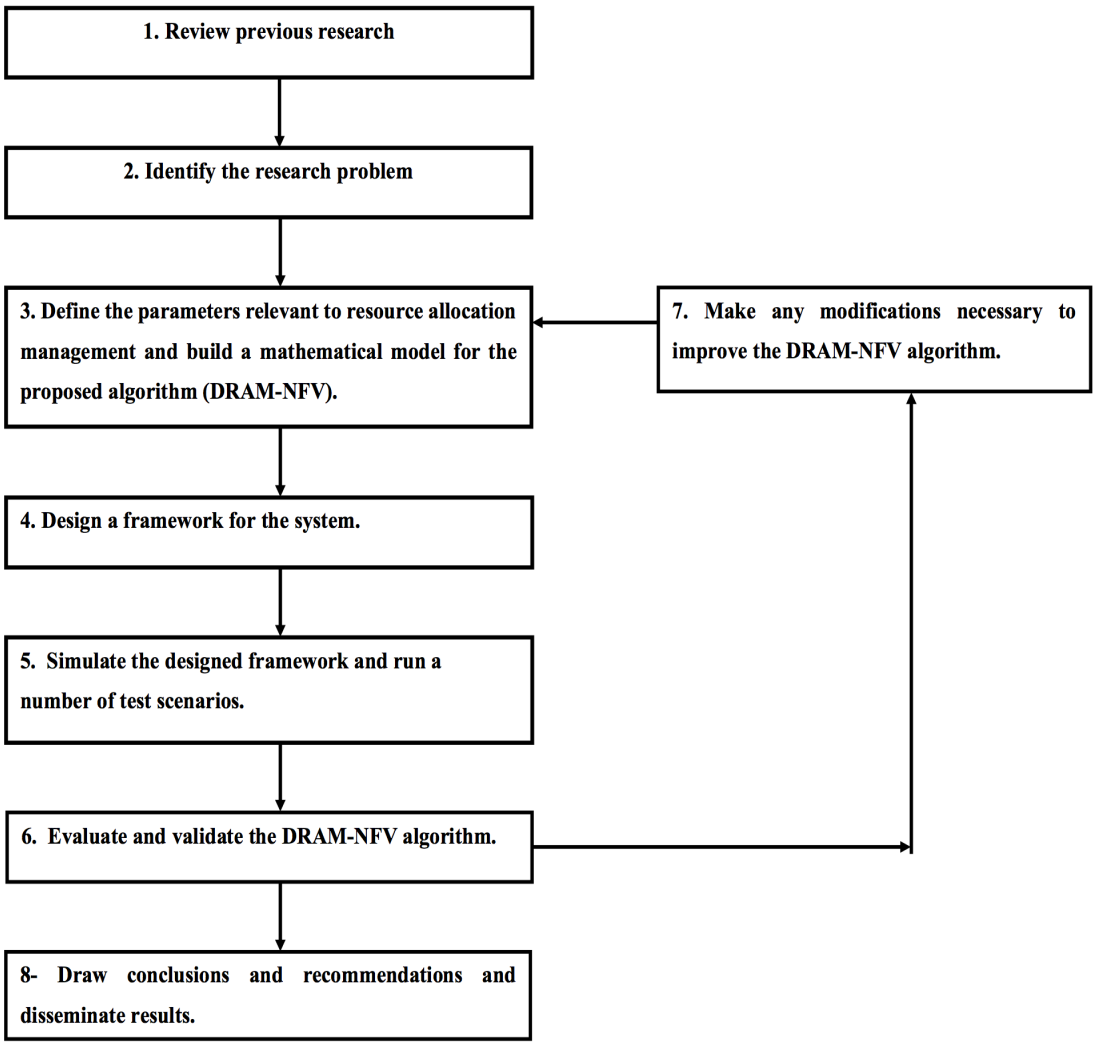


Figure 1-3, Research methodology.

1.9 Thesis Outlines:

The remainder of this thesis is arranged as follows:

- Chapter 2 outlines the proportional delay DiffServ and recent development on networking QoS.
- Chapter 3 consists of the literature review. This places emphasis on resources allocation in proportional delay DiffServ implementations and on recent applications of the SDN and NFV technologies; it also presents the contribution of the present research.
- Chapter 4 explains the DRAM-NFV algorithm principle and mathematical model.
- Chapter 5 describes the prototype of the DRAM-NFV algorithm which was implemented within a simulation environment (Network Simulator - NS3).
- The results, and the validation and evaluation of the DRAM-NFV algorithm are addressed in Chapter 6.
- Finally, conclusions and recommendations for further work are presented in Chapter 7.

Chapter Two
Background on Proportional Delay DiffServ and Recent Development in Quality of Service

2.1 Introduction:

This chapter provides an overview of the Proportional Delay DiffServ (PDD) technology and other recent technologies which have become prevalent and can be used to achieve Quality of Service (QoS), viz: Network Virtualisation, Software Defined Networks (SDNs), and Network Function Virtualisation (NFV).

The DiffServ architecture allows both absolute (quantitative) and relative (qualitative) service differentiation. The absolute service differentiation aims to provide a service class with a quantitative performance level (minimum forwarding rate, or maximum loss rate) at certain links or network paths. On the other hand, the relative differentiation service provides a number of classes with preferential performance. The higher classes will provide better QoS to flows than the lower classes will (Dovrolis & Ramanathan 1999) – in terms of both queuing delays and packet losses. However, the exact QoS for each class is not specified and depends on the traffic conditions and the DiffServ mechanisms that the network deploys.

There are four different models which can be used in regard to the relative delay differentiation approach (Dovrolis & Ramanathan 1999). First, there is price differentiation which is based on an appropriate pricing scheme such that higher classes are more expensive. This is considered the simplest relative differentiation model and it is informed by the assumption that "*higher prices will lead to lower loads in the higher classes*". Such a differentiation cannot always provide consistent class differentiation because when the higher classes get overloaded, they will, in fact, offer worse packet forwarding than the lower classes. Consequently, the relative QoS differentiation between classes varies with the class loads. Capacity differentiation is another approach to achieving relative differentiation. In this approach, a larger amount of forwarding resources, in terms of bandwidth or buffer space, are allocated to the higher classes – relative to the expected loads in each class. Dedicating a larger relative share of the link bandwidth to the higher classes leads to lower average delays for the higher classes. However, increasing the traffic in the higher classes causes these classes to be offered worse packet forwarding than lower classes and increases their average delays so this scheme also cannot provide consistent differentiation between classes. Strict prioritization is another relative

differentiation model. Here, the higher priority classes are serviced first before lower classes. This service scheme provides a consistent differentiation between classes that does not depend on load variations (i.e. higher classes are always better). However, if the higher classes are persistently backlogged, this situation can result in long 'starvation' periods for the lower classes if no restriction is placed on the load of the higher classes. In addition, a strict prioritization scheme does not achieve a controllable differentiation between service classes because such a scheme does not allow the network operator to adjust the relative QoS between classes. Finally, proportional delay differentiation is another type of relative differentiation model and this model will be discussed in the next section.

Throughout the successive sections of this chapter, the standard DiffServ classes and queue management techniques will be illustrated. In addition: we will explain how the Cloud and the SDN technologies participate in supporting the NFV technology.

2.2 Proportional Delay DiffServ (PDD) Model:

In general, the Proportional Differentiated model is a relative differentiation model which provides a controllable and predictable differentiation scheme (Dovrolis & Ramanathan 1999). Controllability means that the quality spacing between classes can be adjusted by the network operator based on policy criteria. Predictability means that the differentiation scheme among classes is consistent (i.e. higher classes are always better or at least no worse than lower classes). These properties must be maintained even across short timescales and independent of the variation in class loads.

The proportional differentiation model states that "*certain class performance metrics (queuing delay or packet losses) should be proportional to the differentiation parameters the network operator chooses*" (Dovrolis 2000). In detail, the scheme can be described as follows. Suppose that $\overline{M}_i(t, t+\tau)$ is a performance measure for class (i) in time interval $(t, t+\tau)$, where τ>0; it represents the monitoring timescale (the value of τ should be relatively small to achieve differentiation over a short timescale). The model imposes constraints as illustrated in Equation 2-1 for all pairs of classes and for all time intervals $(t, t+\tau)$ in which $\overline{M}_i(t, t+\tau)$ and $\overline{M}_j(t, t+\tau)$are defined.

$$\frac{\overline{M}_i(t, t+\tau)}{\overline{M}_j(t, t+\tau)} = \frac{c_i}{c_j}$$

2-1

Where $c_1 < c_2 < \ldots < c_n$ are the generic quality differentiation parameters.

Although the actual quality level of each class in the proportional differentiation model will vary with class loads, the quality ratio between classes will remain in place and controllable by the network operator, independent of class loads.

The proportional differentiation model can be applied in the context of queuing delays (Dovrolis 2000) by setting $\overline{M}_i(t, t+\tau) = \frac{1}{\bar{d}_i(t,t+\tau)}$ in Equation 2-1, where $\bar{d}_i(t, t+\tau)$ is the average queuing delay of the class (i) packets that depart in the time interval$(t, t+\tau)$. If there are no packets, $(t, t+\tau)$ is not defined.

The proportional delay differentiation model states that "for all pairs of classes and for all time intervals $(t, t+\tau)$ in which both $\bar{d}_i(t, t+\tau)$ and $\bar{d}_j(t, t+\tau)$ are defined, then:

$$\frac{\bar{d}_i(t, t+\tau)}{\bar{d}_j(t, t+\tau)} = \frac{\delta_i}{\delta_j}$$

2-2

where the parameters δ in Equation 2-2 are the Delay Differentiation Parameters (DDP), ordered as $\delta_1 > \delta_2 > \cdots > \delta_N$, that are configured by the domain administrator. The proportional delay differentiation model is used as the differentiation model for this research.

2.2.1 Standard Differentiated Service Classes:

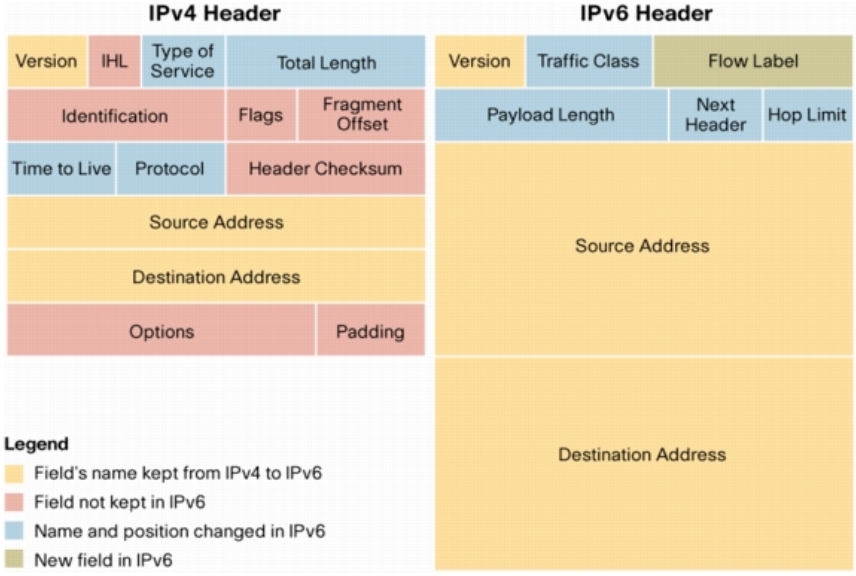
The DiffServ service class is specified in each IP packet via a short label (eight bits) in IPv4 or IPv6 headers (it is called as Type of Service in IPv4 and as Traffic Class in IPv6). Figure 2-1 shows the IPv4 and IPv6 headers fields (Cisco Systems 2006). This mentioned label is also called the (DSCP). In fact, just six bits of the DiffServ field constitute the (DSCP) (Nichols et al. 1998). This value identifies a processing action to be performed by routers on the incoming packet called a per hop behaviour (PHB) (Blake et al. 1998) (Carpenter & Faucheur 2000). Figure 2-2 and Figure 2-3 show the DiffServ field in an IP header and a DSCP allocation table (WANG 2001) respectively.

The (IETF) identified three types of differentiated forwarding service:

- Expedited Forwarding (EF) (Nichols et al. 1999) provides minimal delay, jitter and packet loss and guarantees the required bandwidth. The (EF) service is suitable for delay-sensitive applications such as voice and video. The expedited packet queue is given priority over the regular one via the use of a priority scheduler. Expedited packets see a network as unloaded, even when there is, in fact, a heavy load of regular traffic (Tanenbaum & Wetherall 2011).
- Assured Forwarding (AF) (Heinanen et al. 1999) classifies IP packets into four traffic classes and three levels of drop precedence (low, medium, and high). In the case of congestion,

high-drop-precedence packets are more likely to be dropped than low-drop-precedence packets. These two factors (the traffic classes and the drop precedences) together define 12 service classes. The implementation of an Assured Forwarding service requires an active queue management algorithm capable of solving possible long-term congestion problems.

- Best-Effort (BE) forwarding represents the simplest type of service that a network can offer; it does not provide any form of resource assurance to traffic flows (WANG 2001). The network treats all packets equally. When a link is congested, packets are simply pushed out as the queue overflows. A best effort (BE) service is appropriate for some applications that can tolerate large delay variations and packet losses, such as file transfer and email.



Field	Description	Field	Description
Version	Indicate the version of IP currently used.	Time-to-Live (IPv6 : Hop Limit)	Maintains a counter that gradually decrements down to zero, at which point the datagram is discarded.
IHL (IP-Header Length)	Indicates the datagram header length in 32 bits words	Protocol (IPv6: Next Header)	Indicates which upper-layer protocol receives incoming packets after IP processing is complete
Type of Service (IPv6: Traffic Class)	Differentiated Services Code Point (DSCP) – 6 bits	Header Checksum	Helps ensure IP header integrity.
Total Length (IPv6: Payload Length)	Specifies the length, in bytes, of the entire IP packet, including the data and header.	Source Address	Specifies the sending node.
Identification	Identifies the current datagram.	Destination Address	Specifies the receiving node.
Flags	Consists of a 3-bit field of which the two low-order (least-significant) bits control fragmentation. The low-order bit specifies whether the packet can be fragmented. The middle bit specifies whether the packet is the last fragment in a series of fragmented packets. The third or high-order bit is not used.	Options	Allows IP to support various options, such as security.
		Flow Label (IPv6 only)	20 bits, originally created for giving real-time applications special service
Fragment Offset	Indicates the position of the fragment's data relative to the beginning of the data in the original datagram, which allows the destination IP process to properly reconstruct the original datagram.	Data (variable)	Contains upper-layer information.

Figure 2-1, IPv4 and IPv6 headers fields.

Precedence	D	T	R	0	0

Differentiated Service Field in IP Header

Field Definition	
Bit	Description
0-2	Precedence
3	0 = Normal Delay
	1 = Low Delay
4	0 = Normal throughput
	1 = High throughput
5	0 = Normal reliability
	1 = High reliability
6-7	Reserved for future use

Application	Bandwidth	Delay	Jitter	Loss
Email	**Low**	**Low**	**Low**	**Medium**
File Sharing	**High**	**Low**	**Low**	**Medium**
Web Access	**Medium**	**Medium**	**Low**	**Medium**
Remote Login	**Low**	**Medium**	**Medium**	**Medium**
Audio on Demand	**Low**	**Low**	**High**	**Low**
Video on Demand	**High**	**Low**	**High**	**Low**
Telephony	**Low**	**High**	**High**	**Low**
Videoconferencing	**High**	**High**	**High**	**Low**

Figure 2-2, DiffServ field in IP header.

DSCP	PHB	DSCP	PHB
000000	CS0(DE)	100000	CS4
000001	EXP/LU	100001	EXP/LU
000010	-	100010	AF41
000011	EXP/LU	100011	EXP/LU
000100	-	100100	AF42
000101	EXP/LU	100101	EXP/LU
000110	-	100110	AF43
000111	EXP/LU	100111	EXP/LU
001000	CS1	101000	CS5
001001	EXP/LU	101001	EXP/LU
001010	AF11	101010	-
001011	EXP/LU	101011	EXP/LU
001100	AF12	101100	-
001101	EXP/LU	101101	EXP/LU
001110	AF13	101110	EF
001111	EXP/LU	101111	EXP/LU
010000	CS2	110000	CS6
010001	EXP/LU	110001	EXP/LU
010010	AF21	110010	-
010011	EXP/LU	110011	EXP/LU
010100	AF22	110100	-
010101	EXP/LU	110101	EXP/LU
010110	AF23	110110	-
010111	EXP/LU	110111	EXP/LU
011000	CS3	111000	CS7
011001	EXP/LU	111001	EXP/LU
011010	AF31	111010	-
011011	EXP/LU	111011	EXP/LU
011100	AF32	111100	-
011101	EXP/LU	111101	EXP/LU
011110	AF33	111110	-
011111	EXP/LU	111111	EXP/LU

AF_{xy}: Assured Forwarding PHB (Class *x*, drop precedence *y*).

CS*x*: Class selector PHB *x*.

DE: Default PHB (CS0).

EF: Expedited Forwarding PHB.

EXP/LU: Experimental/ Local use.

Figure 2-3, DSCP allocation table.

2.2.2 Queue Management Techniques for Differentiated Services:

Many packet drop algorithms have been proposed. Tail drop is a simple passive queue management technique. All the packets arriving at the buffer are dropped indiscriminately when the buffer is full (Bodin & Schelcn 2001). This method cannot provide any control of congestion; the congested situation tends to go into an oscillation between peaked congestion and off-congestion (Kun I. Park 2005). The Random Early Detection Algorithm (RED) is used in a router to detect congestion and to control it by dropping packets early in congestion intervals. Here, the average queue size for a particular period in the router is computed and when this average queue size exceeds a certain threshold, arriving packets are dropped with a particular probability that is a function of the average queue size. A high average queue size mean that the probability of packets being dropped is high (Clark & Fang 1998).

The Weighted Random Early Detection algorithm (WRED) is the same as the RED algorithm, but with multiple drop profiles. In WRED, rather than using a single drop profile for all queues, different drop profiles may be defined for individual queues. In addition, multiple drop profiles may be defined within a single queue (Kun I. Park 2005). WRED is the algorithm used in this research.

2.3 Network Virtualisation, SDN and NFV:

Network virtualization in the Internet can be described as a networking technology that creates dedicated Virtual Networks (VNs) over a physical infrastructure (Han et al. 2016). It can also be described as a networking environment which allows one or multiple service providers to form heterogeneous virtual networks that co-exit together but in isolation from each other and to deploy customized end-to-end services on those virtual networks by effectively sharing and utilizing the underlying network resources furnished by the infrastructure providers (Chowdhury & Boutaba 2009). The underlying network resources (the physical network infrastructure) consist of physical links and nodes that are virtualized and made available to virtual networks. Figure 2-4 illustrates a network virtualization environment (Duan et al. 2012). With network virtualization, network services provisioning is decoupled from the data transportation mechanisms.

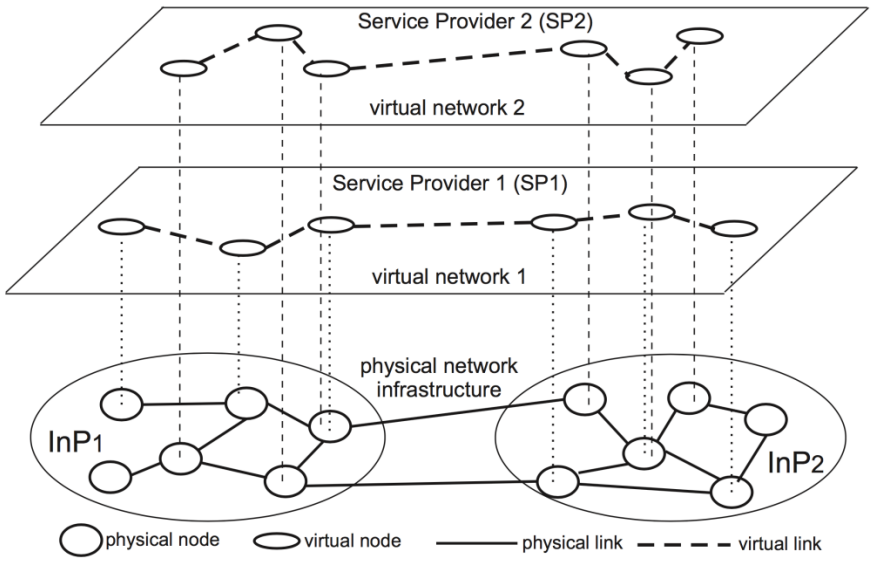


Figure 2-4, Illustration of a network virtualization environment.

Cloud computing refers to both applications and hardware (storage and processing) that can be delivered as virtual services over the Internet (Armbrust et al. 2009) (Mell & Grance 2011). It can also be defined as a large scale distributed computing paradigm in which a pool of

abstracted, virtualized, dynamically scalable computing functions and services are delivered on demand to external customers over the Internet (Foster et al. 2008).

There are four main classifications of Cloud services (Dong Xu 2010).

i. Software-as-a-Service (SaaS) is a Cloud based service model whereby applications reside on the Cloud and are offered to end-users. The application's services can be provided as utilities to its customers.

ii. Platform-as-a-Service (PaaS) is a Cloud based service model used by Cloud computing developers to implement and deploy their applications onto the cloud.

iii. Infrastructure-as-a-Service (IaaS) or Hardware as a service (HaaS) is a Cloud based service model. It provides computing resources in terms of virtual hardware (storage and processing) (Buyya et al. 2013) (Hill et al. 2013) (Ashon & Ilyas 2011). The virtual hardware services can be provided as utilities to its customers.

iv. Network as a Service (NaaS): is a Cloud based service model that offers network connectivity services virtually over the Internet. The network connectivity services can be provided as utilities to its customers (M.P.V. Manthena 2015).

Networking plays an important role in Cloud computing because all Cloud services represent the remote delivery of computing resources via the Internet. Networking also provides data communications within Cloud data centres and among such data centres distributed at different locations. So, Cloud services consist of computing functions (storage and processing) that are provided by the Cloud infrastructure and communications functions (Networking) offered by the Internet. The convergence of networking and Cloud computing through the virtualization of networking and computing resources allows for a Network-as-a-Service (NaaS) paradigm. The services delivered to end users using the NaaS paradigm include computing services provided by the Cloud infrastructure and network services offered by the network infrastructure. Figure 2-5 shows a NaaS based framework for network–Cloud convergence.

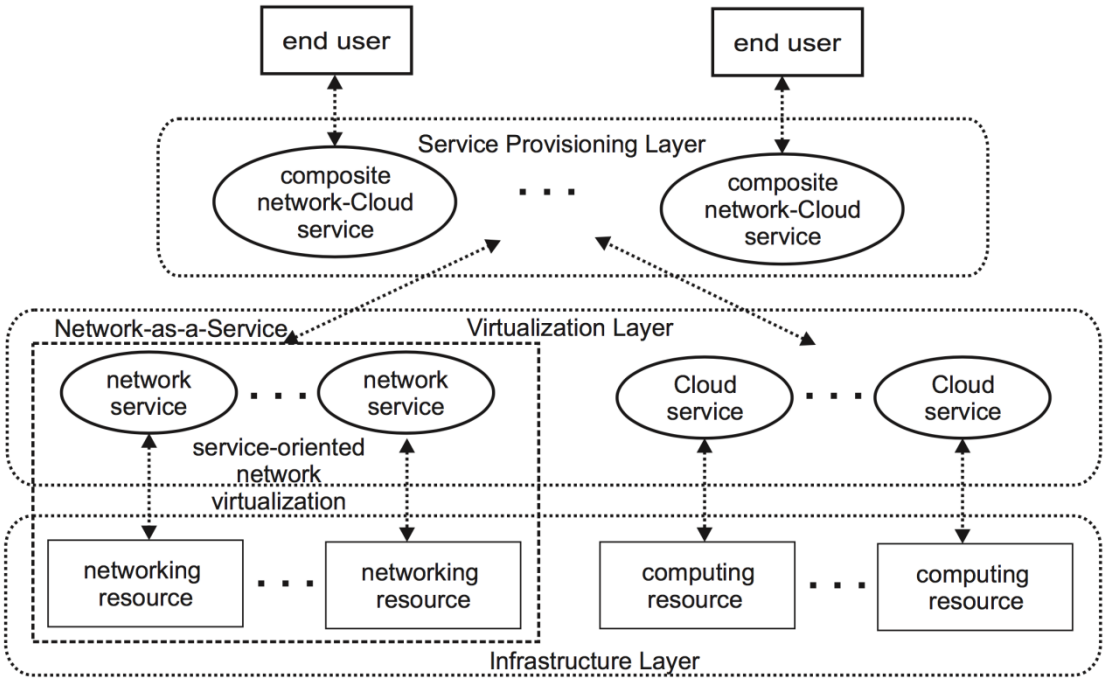


Figure 2-5, NaaS based framework for network cloud convergence.

SDN is a networking model that achieves network virtualisation. It offers an opportunity to loosen the limitations inherent in conventional network infrastructures in which the network operators need to configure each individual network device separately in order to implement the desired network policies because the control plane and the data plane are bundled inside the networking devices (Theophilus Benson 2009)**.** SDN was originally defined by a research team at Stanford University (Kate Greene 2009) (McKeown et al. 2008) as "a network architecture where the forwarding state in the data plane is managed by a remote-control plane decoupled from the data plane". In an SDN context, the forwarding decisions are flow based, rather than destination based. The flow abstraction allows for the unification of the behaviours of different types of network devices, including routers, switches, firewalls and so on (Jamjoom et al. 2014). As a result of this new infrastructure, network switches and routers become simple forwarding devices and the control logic is implemented in a logically centralized controller. Thus, the processes of policy enforcement and network (re)configuration and evolution become much simpler (Kim & Feamster 2013) and a network environment can become capable of dynamically dealing with faults and load changes (Raghavan et al. 2012) (Kim & Feamster 2013).

The control plane in SDN is considered to be the core of the SDN; it manages and monitors the SDN domain by involving one or more centralized SDN controllers. An SDN controller or Network Operating System (NOS) is a piece of software that runs on commodity server

technology. It takes requests from the Management plane (Management Application) via a well-defined (API) and accomplishes consolidated management and monitoring of forwarding devices in data plane via another well-defined (API).

There are two control models which can be used in an SDN (Rao 2014) (Kreutz et al. 2015). These are the centralized controller model, whereby a single entity manages all SDN forwarding devices; and the distributed controller model, which controls a cluster of nodes and the distributed controllers are connected to each other using also a well-defined (APIs) to import/export data. The distributed controller model is more scalable and dependable than the centralized model.

The Management plane in SDN is an array of network services applications run separately and directly on the controller within each SDN domain (McKeown et al. 2008).

The Network Functions or Services (NF) can be divided into control plane functionalities and data plane functionalities. Control plane functionalities are responsible for controlling the network operations: e.g., resources management and controlling network access. Data plane functionalities are responsible for forwarding the traffic (Soares et al. 2014).

The European Telecommunications Standards Institute (ETSI) initiated a new technology called NFV. It exploits the virtualization technique in order to separate network functions from the underlying hardware equipment's, through software applications running on high-volume servers (Cloud). Virtualized network functions can be instantiated in various locations, datacentres or network nodes (Chiosi et al. 2012) (ETSI). NFV reduces network Capital Expenditure (CAPEX), Operating Expenses (OPEX) and complexity by reducing equipment costs and power consumption.

NFV can be considered a complementary technology to SDN and can be combined with it to achieve greater efficiency (Hawilo et al. 2014) (Chen et al. 2015) (Rao 2014) (Han et al. 2015). High volume servers (Cloud) hold multiple (VMs) on which multiple Network Functions (NFs) run. A virtual machine is a software application which can be used to provide the physical computing and networking resources of a Cloud as virtual resources to the SDN network operating system (controller) which is set up on a virtual machine. Each VM represents an SDN controller and these controllers communicate with one another. Consequently, the virtualized network functions can be deployed with less hardware as the virtual machines can be deployed across the system resulting in an increase in system efficiency and flexibility.

With the use of the SDN and NFV technologies, algorithms can be applied flexibly and bring intelligence to the system because, firstly, the SDN controller maintains the network state and the resource allocation and produces the computing results for an optimized network status.

Secondly, NFV provides the infrastructure on which the SDN controller applications run and also provides connections among the virtualized network functions through the connection of the SDN controllers.

The selection of which NFs to virtualize depends on: (A) from a technical perspective, whether the selected NF can actually be virtualized or not (e.g., the issues might be the availability of hardware and performance limitations); and (B) from an economic and operational perspective, will the effort of virtualizing the NF bring real value? (Soares et al. 2014).

2.4 Chapter Summary:

The Proportional Delay DiffServ (PDD) model is a relative differentiation service model. It provides a number of service classes with preferential performance such that the higher classes receive a better QoS in terms of both queuing delays and packet losses. The exact QoS for each service class in this model is not specified but depends on the traffic conditions in the DiffServ router (the per hop behaviour).

Network Function Virtualisation (NFV) aims to implement network services or functions via software elements executed on a virtualisation infrastructure. The Network-as-a-Service (NaaS) Cloud based service model constitutes the infrastructure for an NFV. It provides the physical computing and networking resources of a Cloud environment as virtualised services to the network function that requires to be virtualised.

The Software Defined Network (SDN) model is a networking model that provides support for network virtualisation. In SDN, the control layer (Controller) of a forwarding device is separated from the data layer. SDN makes the data layer of physical network devices (forwarding devices) programmable remotely and accomplishes the consolidated management and monitoring of forwarding devices in relation to the requests that are received from the management application of a network function (the Management layer).

The SDN and NFV technologies can be used to implement a new model for Proportional Delay DiffServ systems. In this model, managing the service classes' bandwidth resources at the edge routers of the DiffServ domains is selected as a network function which must be virtualised using the concept of NFV for a number of reasons. Firstly, only some information about the traffic conditions (service class queue lengths and delays) in the edge routers of the DiffServ domains need be sent to the NFV in order to configure resources while the SDN forwarding devices keep forwarding the service class packets according to the configuration settings that are received from the (NFV) management application through their controllers. Secondly, this process of configuring resources via the NFV cannot cause extra delays, especially if there are

enough virtualised computing and networking resources to accommodate the necessary computing and communicating processes. The edge routers of DiffServ domains in this model should be SDN devices and their controllers are set up as software on the Cloud. The controllers will communicate to each other in order to import and export information concerning traffic congestion in any of the service class at the edge routers of the DiffServ domain. They accomplish dynamic consolidated management for the service class bandwidth resources at the edge routers within and between PDD domains, rather than isolated management, because they monitor the traffic conditions in the service classes at the SDN forwarding devices (the edge routers) of the DiffServ domains. The virtualised function for managing service class bandwidth resources takes into consideration the congestion level in the service classes at the ingress router of the downstream DiffServ domain and the traffic conditions at the egress router of the upstream DiffServ domain - to manage service class resources across both domains.

Chapter Three

Literature Review Relating to Proportional Delay DiffServ Resource Management and the Recent Uses of SDN and NFV

3.1 Introduction:

This chapter focuses on the packet scheduling algorithms that can be used to implement the Proportional Delay Differentiated PDD model. The PDD model is achieved by providing a lower delay for the highest priority service class as compared to other classes in long and short timescales, and also by approximating the ratios of average service class delays to the ratios specified by their corresponding differentiated delay parameters. Within the scope of the PDD model, resources can be allocated in a number of ways. They can be allocated based on the normalized average class delay, on the normalized packet delay at the head of the service class queue as in Proportional Average Delay PAD, Waiting Time Priority WTP and Hybrid Proportional Delay HPD scheduling algorithms - or they can be allocated by providing each class with a minimum bandwidth share of the router out-link's capacity, as in Link Sharing Scheduling algorithms and Backlogged Proportional Rate scheduling algorithms.

Some recent contributions related to the NFV and SDN technologies are also presented in this chapter. In addition, the contribution of this present study is presented in this chapter. This latter contribution centres on the use of NFV and SDN technologies to improve the performance of the Dynamic Weighted Fair Queuing (DWFQ) algorithm in allocating resources within and across different DiffServ domains. The reasons for selecting the DWFQ algorithm as opposed to other scheduling algorithms are explained in the last section of this chapter.

3.2 Resources Allocation in Proportional Delay DiffServ domains:

In this section, the scheduling algorithms that are used to implement the PDD model are discussed. A good scheduler for the PDD model should not only achieve or closely approximate the constraints specified by the PDD model, as per Equation 3-1, but it should also provide predictable delay differentiation within short timescales (Dovrolis et al. 2002). This latter desideratum simply means that the performance of higher classes should be better

than the performance of lower classes independent of the aggregate load, the class load distribution, and the timescales in which the performance is measured.

$$\frac{\bar{d}_i}{\bar{d}_j} = \frac{\delta_i}{\delta_j}$$

3-1

Proportional Average Delay (PAD) scheduling (Dovrolis et al. 2002) maintains the average delay $d_i(t)$ for each service class *i*, normalized by the corresponding DDP (δ_i), as illustrated in Equation 3-2. A packet is chosen for forwarding at time t from the backlogged class with the maximum normalized average delay.

$$\tilde{d}_i(t) = {}^{d_i(t)}/_{\delta_i}$$

3-2

This algorithm aims to equalize the normalized average delays among all classes in long term timescales through reducing the differences among their normalized average delays. (Dovrolis et al. 2002) considered the PAD algorithm to be an excellent scheduler in terms of meeting the constraints of the PDD model when certain conditions are met with regard to selecting the DDPs. (Dovrolis et al. 2002) also examined the behaviour of the PAD scheduling algorithm by comparing the ratios of short term average delays between classes and the ratios of DDPs between classes. He found that this scheduling algorithm does not take into account the waiting times of the backlogged packets, and it occasionally allows higher classes to experience much larger queuing delays than their long-term average and the queuing delays of lower classes. Thus, the PAD algorithm does not achieve the predictability criterion required by the PDD model.

Waiting Time Priority (WTP) scheduling was another scheduling algorithm which was examined by (Dovrolis et al. 2002). In WTP scheduling, a packet is assigned a priority that increases proportionally to the packet's waiting time. The higher classes in the WTP algorithm have larger priority increase factors and the packet with the highest priority is serviced first in non-pre-emptive order. WTP maintains the waiting time $w_i(t)$ of the packet at the head of each service class *i*, normalized by the corresponding DDP (δ_i) as illustrated in Equation 3-3.

$$\tilde{w}_i(t) = {}^{w_i(t)}/_{\delta_i}$$

3-3

Whereas the PAD scheduler chooses for service the class with the maximum normalized average delay, WTP chooses for service the class with the maximum normalized head waiting time: i.e., a packet is sent from that backlogged class with the maximum normalised head waiting time. In this manner, WTP attempts also to minimize the differences between the normalized waiting times of the successively departing packets. (Dovrolis et al. 2002) demonstrated the behaviour of the WTP scheduler. That study found that WTP achieves a proportional delay differentiation between successive packet departures, especially when the delays are sufficiently large. Also, (Dovrolis et al. 2002) tested the performance of WTP in short timescales. It was found that the WTP is an excellent scheduler in terms of providing higher classes with lower delays in short timescales and thus, this scheme provides a predictable delay differentiation. Moreover, it also approximates to the PDD model in heavy load conditions as the utilization tends to 100% and the aggregate backlog of classes increases. For lower load conditions, (Dovrolis et al. 2002) found that the WTP algorithm deviates, technically, from the PDD model.

Hybrid Proportional Delay (HPD) scheduling is another scheduling algorithm which was presented by (Dovrolis et al. 2002). This scheduling algorithm combines the methods of the PAD and WTP algorithms in order to create a scheduler that is approximate to the PDD model, when certain conditions are met in relation to selecting DDPs, and that provides a predictable delay differentiation in short timescales. HPD maintains a delay metric for each service class *i*, normalized by the corresponding DDP (δ_i). The delay metric for each class includes the normalized average delay $\tilde{d}_i(t)$ for a service class *I* and the normalized head waiting time, $\tilde{w}_i(t)$, for a service class *I,* as illustrated in Equation 3-4,

$$\tilde{h}_i(t) = g\tilde{d}_i(t) + (1-g)\tilde{w}_i(t) \qquad 3\text{-}4$$

where:

$\tilde{h}_i(t)$ is the normalized hybrid delay for a service class *i*.

$\tilde{d}_i(t)$ is the normalized average delay for a service class *i*.

$\tilde{w}_i(t)$ is the normalized waiting time of the packet at the head of service class *i*.

g is known as the HPD parameter $0 \leq g \leq 1$.

When *g* is zero HPD becomes equivalent to WTP, when *g* is one HPD becomes equivalent to PAD. For other values of *g*, HPD combines the attributes of the PAD and WTP schedulers. (Dovrolis et al. 2002) used a simulator to estimate the best value of *g* (0.875) whereby HPD combines the attributes of the PAD and WTP schedulers.

In contrast to PAD and WTP schedulers, HPD choose for service the class with the maximum normalized hybrid delay: i.e., a packet will be sent from that backlogged class which has the maximum normalised hybrid delay. (Dovrolis et al. 2002) tested the performance of the HPD algorithm and concluded the following.

i. In heavy load conditions, about 90% utilization, all three algorithms, PAD, WTP, and HPD, meet the PDD model criteria: i.e., they keep within the constraints on ratios specified by the PDD model.
ii. In lower load conditions, HPD is closer to the PDD model than WTP but it does not reach the optimal behaviour which is exhibited by PAD.
iii. HPD closely approximates the PDD model independent of class load distribution. The approximate error increases as the utilization decreases and as the delay differentiation between classes becomes more extreme.
iv. HPD does not have the predictability problem that PAD has. This means that the HPD algorithm manages to provide lower delays to higher classes in both short and long term timescales.

Link sharing scheduling algorithms aim to provide each class with a minimum bandwidth share of the link's capacity. Examples of such schedulers include the Generalized Processor Sharing (GPS) scheduler (Heinanen et al. 1999) (Demers et al. 1989) and its approximations such as Class Based Queuing (CBQ) (Floyd & Jacobson 1995), Weighted Fair Queuing (WFQ) (Demers et al. 1989) and Hierarchical Packet Fair Queuing (H-PFQ) (Bennett & Zhang 1997). GPS defines a packet based scheduler that provides a minimum service rate $r_i(t)$ to each backlogged class *I* at time t, as illustrated in Equation 3-5. Such a scheduler (and the equation) takes into account class weights $\{w_j, j = 1 \ldots . . N\}$, where N represents the number of service classes.

$$r_i(t) = C \frac{w_i}{\sum_{j \in B(t)} w_j}$$

3-5

where:

C is the link bandwidth (capacity), and

$B(t)$ is the set of backlogged classes at time t.

The weights used by GPS are the performance controls that determine the delay differentiation between classes. If these weights are selected such that one or more classes have a larger share

of the link capacity than other (lower) classes, relative to the class input loads, then higher classes can be expected to encounter lower queuing delays.

The service classes in CBQ, WFQ and H-PFQ schedulers are guaranteed a certain fraction of a link's capacity, sharing any available excess bandwidth. Link sharing scheduling algorithms can be used in the proportional DiffServ model. (Dovrolis 2000) demonstrated that they can provide controllable delay differentiation, but they are too sensitive to class load distribution. Class Based Queuing (CBQ) allows traffic to share bandwidth. It divides traffic into queues and assigns each queue a specific amount of network bandwidth (Floyd & Jacobson 1995).

Weighted Fair Queuing (WFQ) is widely used in QoS routers (Chin-Chang Li et al. 2000), it offers fair queuing that divides the available bandwidth across queues of traffic based on static weights assigned independently to each service class queue (Panza et al. 2006) by the administrator of the domain. The weight of a class can be specified via any QoS parameter – such as service rate or delay. An absolute service class rate can be reserved easily by assigning a fixed weight to a service class queue (Chin-Chang Li et al. 2000).

WFQ is a controllable delay differentiation scheduler wherein the load distribution across service classes is uniform. However, the drawback of WFQ is that slight changes in the class load distribution dramatically affect the resulting delay differentiation and can cause a priority inversion between the offered service classes. This variation in differentiation and inversion of priority are not acceptable for differentiated services networks. (Dovrolis 2000) suggests a solution to this drawback by presenting an algorithm that can adjust the service class queues weights dynamically based on current class loads.

Backlog Proportional Rate (BPR) scheduling is considered to be a dynamic version of the GPS scheduler (Dovrolis et al. 1999). The class weights are dynamically adjusted based on the class loads at any given moment in time. For two backlogged classes *i* and *j*, the service rate allocation used by BPR follows the proportional constraint shown in Equation 3-6. The sum of the assigned service rates $\sum_{i=1}^{N} r_i(t)$ must be equal to the link capacity C when the scheduler is busy.

$$\frac{r_i(t)}{r_j(t)} = \frac{\delta_j}{\delta_i}\frac{q_i(t)}{q_j(t)}$$

3-6

where $q_i(t)$ is the backlog for queue *i* at time t.

(Dovrolis et al. 1999) studied the performance of BPR scheduling. In heavy load conditions, BPR approaches the PDD model asymptotically but it does not do as well as WTP. Moreover,

BPR suffers from a simultaneous queue clearing property such that, after busy periods, all queues controlled by a BPR scheduler become empty at the same time.

(Moret, Yan 1998) and (Chin-Chang Li et al. 2000) have proposed two more proportional delay schedulers called the Proportional Queue Control Mechanism (PQCM) and Dynamic Weighted Fair Queuing (D-WFQ) respectively. Both these scheduler schemes are based on the GPS rate allocation mechanism. They both adjust the GPS weights periodically, every Δ seconds, based on the measured service class queue length as represented by the class backlogs and the input rates. In this manner, they attempt to achieve proportional average delay differentiation in the next time interval of Δ seconds. PQCM uses the backlogs as measured on the instant to adjust the weights while D-WFQ uses exponential average estimators for the computation of the service class queue lengths. The difference between PQCM and D-WFQ schedulers and the BPR scheduler is that BPR adjusts the weights after each packet departure and so is not based on a fixed period Δ while PQCM and D-WFQ adjust their weights periodically. This latter behaviour introduces a feedback loop in the network such that the future service rates depend on the service class queue length over a previous time interval.

(Sharma et al. 2014) has designed a dynamic packet scheduling scheme for DiffServ called Dynamic Benefit Weight Scheduling (DBWS). It is based on the weighted round robin (WRR) policy. DBWS avoids the overbooking of resources and provides a guaranteed service for the Expedited Forwarding class (EF). The weight of the (EF) class is predicted based on the previous (EF) allocated weight at time (t-1) and the estimated average increase in the queue length of the (EF) buffer.

3.3 Contributions of the SDN and NFV Technologies:

(Guck & Kellerer 2014) proposed a model for achieving end-to-end real time QoS with Software Defined Networking through providing a centralized real time communication service.

(Karaman et al. 2015) utilized SDN to give an ensured QoS to premium clients in a VoIP system congested by both VoIP and non-VoIP backlogged traffic through a constraining bandwidth. Their methodology allocated flows to the various available queues and adapted its routing decisions based on network conditions.

(Alipio et al. 2016) implemented a real time testbed for Software Defined Networks in order to demonstrate how the QoS mechanism (Priority Queuing) can be implemented on an OpenFlow testbed. The testbed used Raspberry Pi to implement the SDN forwarding device (OpenFlow switch) and also the L2 forwarding module of an SDN controller (POX) as the basis of the

customized controllers in the testbed implementation. The scheduling algorithm which was implemented for priority queuing manages the flows with respect to the different service classes' QoS requirements and provides priority levels to these different service classes.

(An et al. 2016) used the network simulator NS2 to present a dynamic priority adjustment algorithm which guarantees different delay deadlines for real time flows in SDN based networks by achieving a centralized view of the overall network.

(Hu et al. 2015) proposed a framework for network management and resources allocation which enables current Internet implementations to provide better QoS guarantees for end users and applications. (Hu et al. 2015) built a software defined overlay network, over the IP network, and took advantage of the characteristics of per-flow management in SDN to satisfy the resource demands of these applications. Hu's methodology introduced SDN into the network edge and reconstructed the access network to improve the integration and flexibility of network management and control. Its SDN-based overlay network demonstrated resource scheduling mechanisms which guarantee prescribed levels of QoS and also demonstrated the coupling mechanism which was designed to implement the collaboration between software defined networks and IP networks to achieve better resource utilization. In addition, this overlay network implemented packet classification, flow labels and state maintenance to assist the core network to ensure the QoS of important applications.

(Wibowo et al. 2017) provided a general review of the evolution of multi-domain SDNs and the major challenges which exist for future research. This review analysed the main research issues and approaches relating to the development of multi-domain networks in the future, and it provided an overview of the implementation of a controller which used open source resources as well as commercial systems. This illustrated the differences in controller features.

(Karakus & Durresi 2017) presented a survey of relevant research on the maintenance of QoS using OpenFlow-enabled SDN networks from the point of view of examining the role of SDN. They looked at Multimedia flow routing mechanisms, inter-domain routing mechanisms, resource reservation mechanisms, queue management and scheduling mechanisms, Quality of Experience (QoE) - aware mechanisms, network monitoring mechanisms, and other QoS-centric mechanisms such as virtualization based QoS provisioning and QoS policy management, etc. In addition, they discussed the QoS capabilities of the OpenFlow protocol by reviewing its versions along with some well-known, open-source, and community-driven controller projects. Furthermore, their review outlined the potential challenges and the open questions which need to be addressed further in order to produce better and more complete QoS abilities in SDN/OpenFlow networks.

(Soares et al. 2014) and (M.P.V. Manthena 2015) presented a platform infrastructure architecture for NFV and SDN. Many studies have emerged recently in the telecommunication and computer networks fields that use the cutting-edge techniques of NFV and SDN. (Batalle et al. 2013) presented a design and analysis for the virtualizing of one of the primary network functions (the routing function) that is offered by network routers. (Lee et al. 2016) used the NFV technique to develop a new low latency handover technology for real-time applications in mobile environments – such as mobile cloud access and VoIP. (Volvach & Globa 2016) proposed a methodology, using the SDN and NFV technologies, for the fast recovery of mobile network node functionality in the event of a disaster. (Al-Quzweeni et al. 2016) proposed a framework for improving energy efficiency using NFV in 5G networks. (Luo et al. 2016) proposed advanced algorithms for controllers in industrial wireless sensor networks based on SDN and NFV techniques which also improve energy efficiency. (Urgun & Kavak 2016) discussed the use of SDN and NFV approaches in the QoS mechanism for a cellular core network in terms of traffic classification, traffic monitoring and traffic engineering methods.

The above is a quick overview of some of the most recent research work in the telecommunication and computer networks fields that has made use of the NFV and SDN concepts. There are other studies which have emerged in this field but our focus does not allow us to detail them here.

3.4 The Contribution of this Research:

No previous work has attempted to orchestrate the management of resources across different proportional delay DiffServ domains. The concept of NFV and SDN technologies are used in this research in order to enhance the performance of the DWFQ algorithm (Chin-Chang Li et al. 2000). The contribution of this research is to introduce a new resource management algorithm to allocate resources within and between proportional delay DiffServ domains; this new algorithm is called **"Dynamic Resource Allocation Management – Network Function Virtualization (DRAM-NFV)"**. The DRAM-NFV algorithm manages the resources across different proportional delay differentiated services domains dynamically in the event of congestion in addition to managing the resources dynamically within the edge routers of domains.

A previous work (Chin-Chang Li et al. 2000) studied the performance of the DWFQ scheduling algorithm in achieving a proportional delay differentiation among service classes within a single edge router of a DiffServ domain. In contrast, the work presented here aims to improve the performance of the DWFQ algorithm so that it manages the resources across different

DiffServ domains in the event of congestion in addition to managing resources within the edge routers of the DiffServ domains. There are many differences between the present work and previous studies. In this research, we study the dynamic allocation of resources in a simulated environment that represents multiple inter-related DiffServ domains. We took into consideration the effects of the availability of core routers and the effects of the queue management technique Weighted Random Early Detection (WRED) on resource management. These factors were not considered in the cited previous work. The performance of the DWFQ algorithm in managing resources within a single edge DiffServ router was studied in (Chin-Chang Li et al. 2000) while this present research uses the concept of NFV and SDN technologies to enhance the performance of DWFQ in managing resources across DiffServ domains in the event of congestion, (in addition to managing resources within the edge DiffServ routers). The new algorithm has been implemented and tested, via the network simulator ns3, using a number of test scenarios with various classes and sources of traffic passing through the network elements as illustrated in subsequent chapters.

3.5 Chapter Summary:

Our objectives in this chapter were to define the scheduling algorithms that are used to implement the proportional delay differentiated model and to define some recent applications of the NFV and SDN technologies. The concept of NFV and SDN technologies are used in this research to enhance the performance of the DWFQ algorithm through presenting a new scheduling algorithm called (DRAM-NFV) that allocates dynamically the resources within the edge routers of the DiffServ domain and across different proportional delay DiffServ domains in the event of congestion. The reasons for selecting DWFQ as opposed to other scheduling algorithms are that the DWFQ can be considered as the dynamic version of the WFQ scheduling algorithm which is used widely in QoS DiffServ routers, that DWFQ shares any available excess bandwidth in the router out link among the service classes, and that this algorithm introduces a feedback loop in the DiffServ network such that the service classes rates and weights are assigned periodically in short timescales and based on the traffic conditions in the service class queues of the proportional delay DiffServ router. The next chapter is related to identifying the mathematical model for the DRAM-NFV algorithm and testing the algorithm on different topologies of DiffServ domains, as explained in the subsequent chapters.

Chapter Four
DRAM-NFV Algorithm: Principles and Mathematical Model

4.1 Introduction:

In this chapter, the principle of the DRAM-NFV algorithm to manage resources within and across different Proportional Delay DiffServ domains will be described in the first section of this chapter, and in the second section of this chapter the set of mathematical expressions on which the DRAM-NFV algorithm is based are presented. These mathematical expressions describe the Proportional Delay DiffServ model and its queue performance in term of average queue delay, average queue length, queue scheduling rate and packet forwarding scheme. Finally, a mathematical expression for managing resources across different Proportional Delay DiffServ domains is also presented. The following mathematical principles will be taken into consideration when implementing the simulation of the test scenarios for the DRAM-NFV algorithm (as described in the next chapters).

4.2 Algorithm Principles:

DiffServ networks are composed of edge (ingress and egress) and core routers. The edge routers process the incoming traffic via a sequence of operations and the core routers forward service class traffic. A DiffServ domain is static, meaning that each router or domain allocates its resources locally or separately based on its traffic condition and the domain configuration parameters [Per Hop Behaviour (PHB)]. Once the domain administrator sets the configuration parameters of a domain, these parameters will not change. In addition, the traffic conditions within a DiffServ domain change continuously and randomly and therefore, congestion is possible in the core routers of the DiffServ domain as a result of increasing traffic from the domain edge routers or from adjacent DiffServ domains, as well as when the resources of the domain core routers cannot cope. Traffic congestion means that the DiffServ domain favours certain amounts of traffic while other network traffic will suffer. This congestion is reflected proportionally in the traffic states of the edge routers of the domain and causes further congestion – much like congestion at road junctions does. If congestion occurs at a road junction (i), this will cause additional congestion at a previous junction (i-1), and so on.

Although the DiffServ has Per Hop Behaviour (PHB) and has a static approach of allocating its resources separately per each domain, it would be better if the Differentiated services domains are managing their resources dynamically rather than statically owing to reasons already mentioned. Instead of having a static DiffServ, we can monitor traffic condition in one domain and make changes to managing resources of other domains so we can now calibrate between DiffServ domains in managing resources. This calibration can be achieved through using the SDN and NFV technologies. The SDN switches, for example OpenFlow switches, can be programmed automatically and remotely according to the traffic conditions within or between DiffServ domains and based on the QoS requirements of the Internet applications traffic. The NFV technology can be used to extract and implement virtually a certain function in a DiffServ network, which is resource management function. Using NFV technology, the management process will be deployed across DiffServ domains rather than being deployed locally. It is still locally, if the traffic condition within a DiffServ domain is normal (there is no congestion). Figure 4-1 demonstrates the principle of resource management across DiffServ domains using the (NFV) concept.

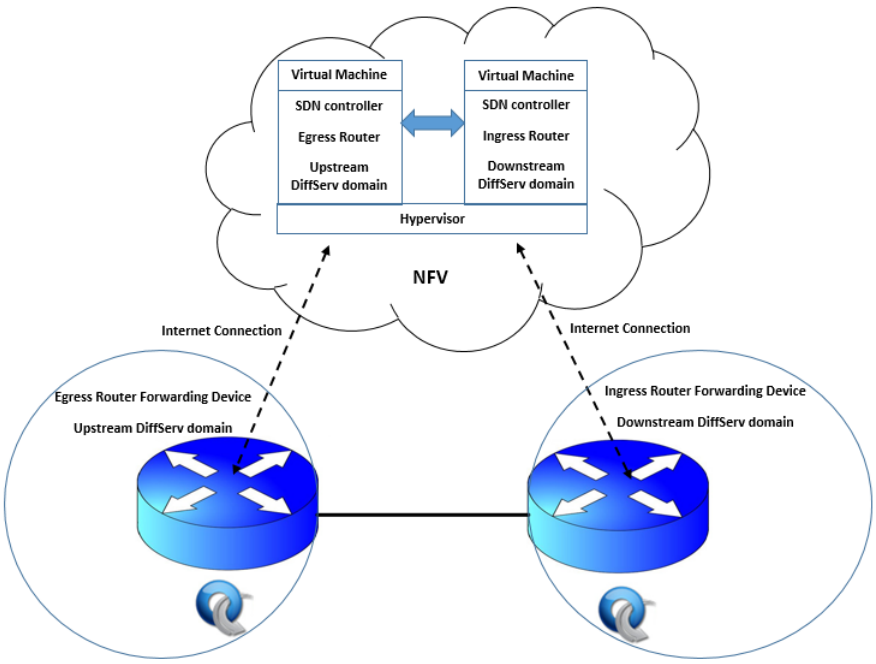


Figure 4-1, Principle of the dynamic resource management algorithm within and between the DiffServ domains using the concept of the NFV.

The edge routers of a DiffServ domain can be represented as OpenFlow switches and are connected to the NFV server (Cloud) via the Internet connection. The DiffServ domain controller is represented as a SDN controller, for example Floodlight, ONOS. The SDN controller is considered as the network operating system of a DiffServ domain over which the network management application runs. The SDN controller sets and manages the queues of OpenFlow switches using the OpenFlow protocol (Open Networking Foundation (ONF) 2012) and OpenFlow Management and Configuration Protocol, OF-CONFIG (Open Networking Foundation (ONF) 2014a). The Floodlight or ONOS controller is a software deployed on the NFV server (Cloud). The Cloud is a high volume physical server and the virtual machines in Figure 4-1 are used to provide the resources of the Cloud, in terms of processing, storage and networking as virtual or shared resources to the SDN controllers of DiffServ domains. Thus, it can be considered that each virtual machine in Figure 4-1 represents an SDN controller for an edge router of a DiffServ domain. The hypervisor in Figure 4-1 is necessary for managing the cloud resources between or among virtual machines. The dynamic resource management process within and between DiffServ domains should occur at regular time intervals, and the network management application concerned with the edge routers of DiffServ domains needs to be informed about the traffic state in the edge router service class queues. Figure 4-2 illustrates the operation of the dynamic resources allocation management algorithm as it is carried out within and across different DiffServ domains using the NFV concept. This management process will request, from the SDN controllers of the edge routers (OpenFlow switches), statistical information about the traffic state in their service class queues. This information will be used to configure the resources among service classes within and between DiffServ domains.

As we have SDN networks, we can monitor and manage the queues and congestion level in DiffServ domains remotely. The network statics of DiffServ domains like average service class queue delay and average service class queue size need to be sent to the NFV server through the DiffServ domain controller in order to achieve a consolidated and dynamic or automatic management for the resources across DiffServ domains. Section 4.3 presents an introduction to using the OpenFlow switches and SDN controller to build a DiffServ module based on the SDN technology.

After sending the network statistics of DiffServ domains to NFV server, the NFV server calculates the congestion levels in each service class at the ingress router of the downstream domain (the domain to which the traffic flow is moving). If there is no congestion, then the domain controller of the upstream DiffServ domain sets the service class weights based on its

configuration parameters. If there is congestion in one of the service classes at the ingress router of the downstream domain, then the NFV server calculates the scheduling rate update factor β and updates the scheduling rate of the equivalent service class at the egress router of the upstream domain.

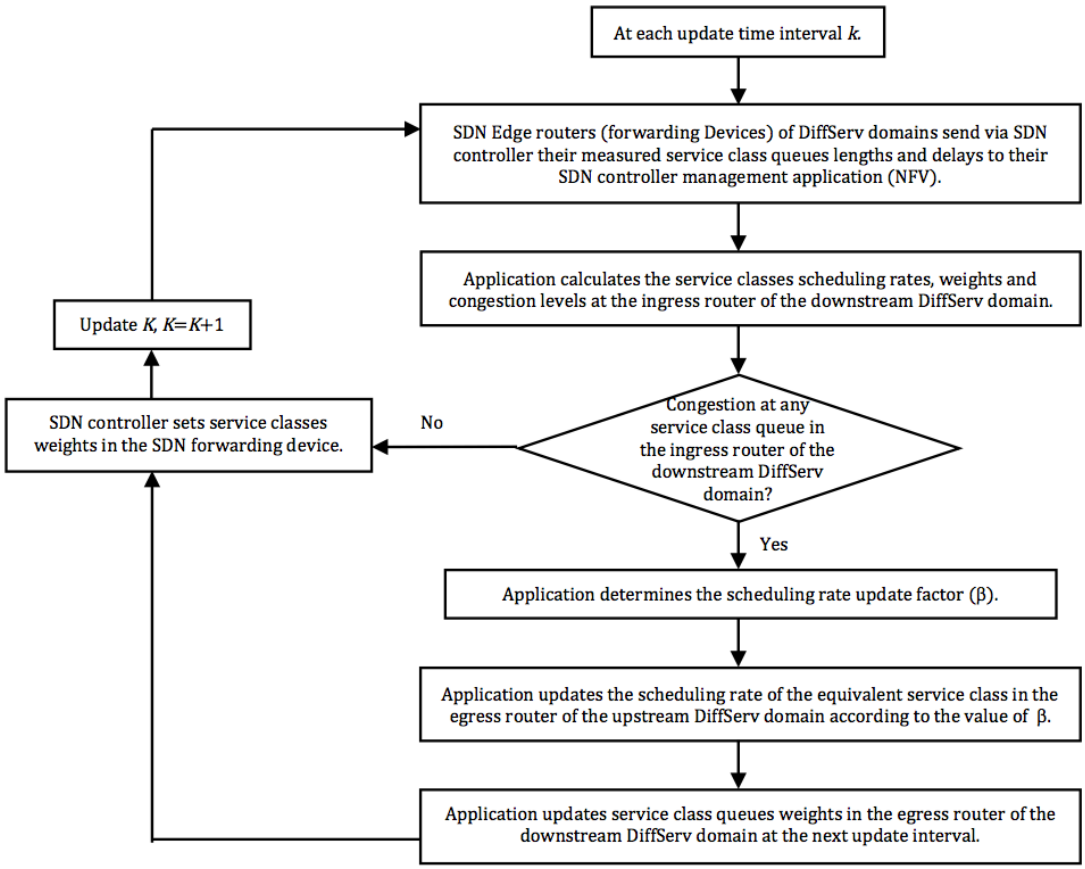


Figure 4-2, flow chart of the dynamic resources allocation management algorithm within and between DiffServ domains using the concept of the NFV.

The condition that I actually decide if the service class is congested or not depends on the depth or size of the service class queue and the low and high threshold values of service class queue. The parameters that we are using in managing resources between DiffServ domains are:

1. $\propto_{i(Low)}$ and $\propto_{i(High)}$, which define the low and high threshold values for the congestion level α_i^{k+1} in service class i at the next update time interval (k+1) based on the low and high threshold values of a service class queue $T_{i(Low)}$ and $T_{i(High)}$ respectively.

2. Average queue depth or size, which is used to measure the congestion level in a service class i (α_i^{k+1}) at the next update time interval (k+1) as a ratio of average service class queue size to service class buffer size.

3. Scheduling rate update factor for service class i at the next update time interval (k+1) (β_i^{k+1}).

Thus, the job of the NFV server is to decide the amount of reduction in the service class scheduling rate between DiffServ domains. Figure 4-3 shows the flow chart for the procedure for managing resources across different DiffServ domains. If the service class i congestion level (α_i^{k+1}) at the ingress router of the downstream domain at the next update time interval (k+1) is less than the low threshold value, then there is no need to reduce the scheduling rate of the equivalent service class queue at the upstream DiffServ domain. It is instead configured as the domain administrator set. If it is between the low and high threshold values $\alpha_{i(\mathrm{Low})} < \alpha_i^{k+1} < \alpha_{i(\mathrm{High})}$, then the scheduling rate is reduced by (β_i^{k+1}) and the relation between β_i^{k+1} and congestion level α_i^{k+1} is linear such that the value of (β_i^{k+1}) is directly proportional to the value of congestion level (α_i^{k+1}). If the service class is very congested, more so than the high threshold value ($\alpha_i^{k+1} > \alpha_{i(\mathrm{High})}$), then ($\beta_i^{k+1}$) is set to the maximum value $\beta_{\max.}$ in order to dampen the service class throughput at the egress router of the upstream domain. The presented algorithm, which is run on the NFV server and uses the SDN technology, takes into consideration all the possibilities of congestion occurring in a service class queue.

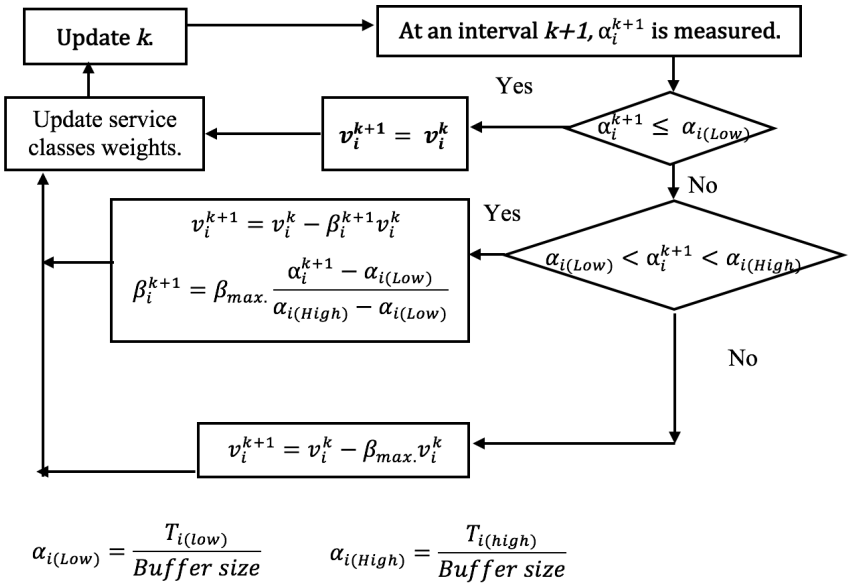


Figure 4-3, Flow chart of illustrating the procedure of managing resources between DiffServ domains.

Managing the resources across DiffServ domains can cause a few milliseconds latency delay owing to the information related to the traffic state being sent from OpenFlow switches to NFV server for processing and sending back the configuration information to configure the queues of OpenFlow switches. However, this delay can be trivial if the NFV server virtual resources which are allocated for the SDN controllers are enough for processing this information and

configuring the queues quickly. If the latency delay exceeds hundreds of milliseconds, this might cause a problem in the performance of a DiffServ domain through increasing the average End to End Delay of the application traffic and affecting the throughput of service classes. The effect of this mentioned delay is not considered in this study because of the limitation of the Network Simulator, NS3 which is used in implementing a prototype of the DRAM-NFV algorithm. Studying the effect of the latency delay, as a future work, needs to provide a physical infrastructure for implementing DRAM-NFV algorithm practically and deserves further attention.

The sliding window protocol in TCP flow ensures that a sender is not flooding a receiver by sending more packets when the receive buffer is already full, as the receiver would not be able to handle them and would need to drop these packets (Tanenbaum & Wetherall 2011). When the UDP traffic increases in a DiffServ domain (downstream domain), the DRAM-NFV algorithm reduces its resources at the egress router of the upstream domain and allocates extra resources for TCP traffic at that router. Consequently, the window size of the sliding window protocol of TCP flow and its throughput will increase and the number of TCP dropped packets will decrease.

The dynamic processes used for calculating the service class average queue lengths, service class average queue delays, service class scheduling rates, service class weights, and for measuring the service class congestion levels, and managing the service class resources across DiffServ domains will be illustrated in detail in the next chapter.

4.3 OpenFlow Switches and SDN Controller:

The Open Network Foundation (ONF) divides the SDN architecture into three major planes, as shown in Figure 4-4. These are (Open Networking Foundation (ONF) 2014b):

- Data Plane: It consists of network devices such as routers, physical/virtual switches, access points etc. These devices are accessible and managed or programmed by SDN controller(s) through an interface called Controller-Data Plane Interfaces, (C-DPI). This interface is used to facilitate the dynamic configuration of OpenFlow switches in a consolidated manner. OpenFlow protocol (McKeown et al. 2008) (Vaughan-Nichols 2011) is the most common standard C-DPI used for communication between controller and a data plane device. It provides the SDN controller with information related to OpenFlow switches, such as flow statistics, and to any changes in device links or ports.

- Controller Plane: It comprises one or more software based SDN controller(s) to provide control functionality by monitoring the network forwarding behaviour through C-DPI. It has interfaces to enable communication among controllers using Intermediate-Controller Plane Interface, I-CPI (Lin et al. 2015), between controllers and network devices using C-DPI and between controllers and applications using Application-Controller Plane Interface, A-CPI for network security or management.

  A controller consists of two essential components: Functional components (Coordinator, Virtualizer etc.) to manage controller behaviour. The other component is the control logic, which maps networking requirements of applications into instructions for network element resources (Open Networking Foundation (ONF) 2014b).

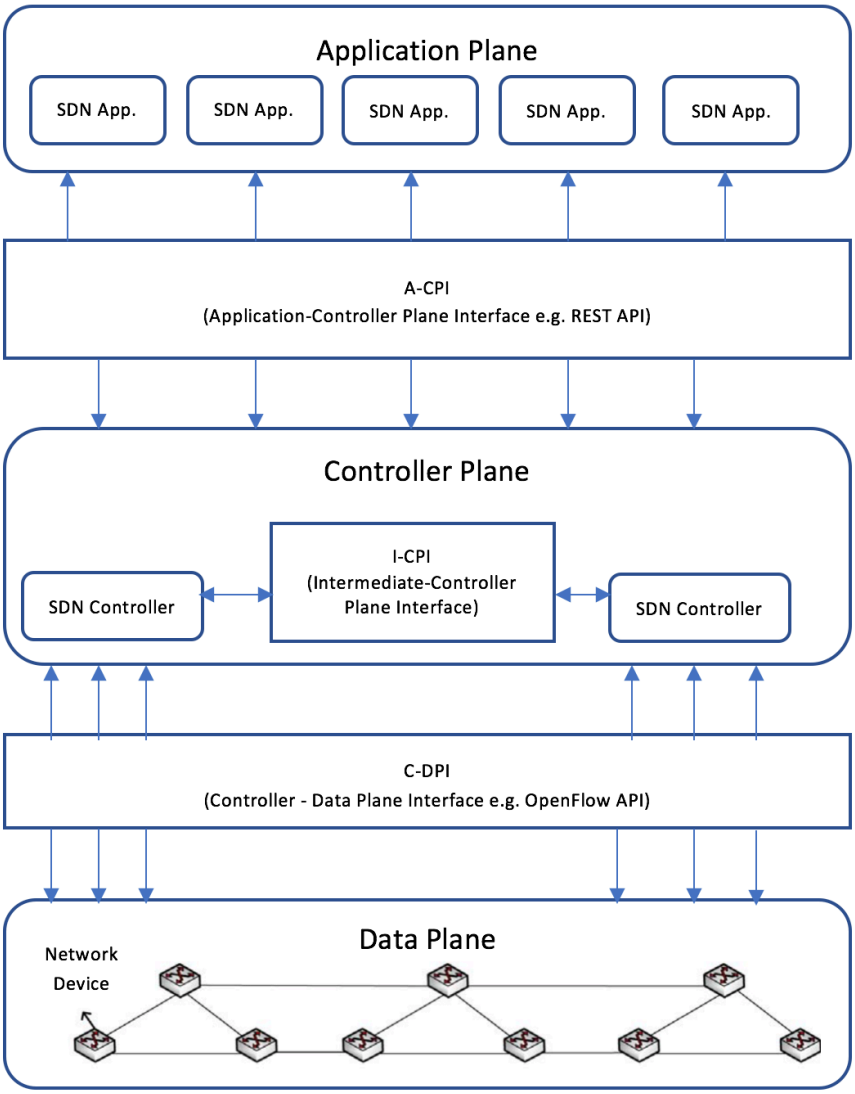


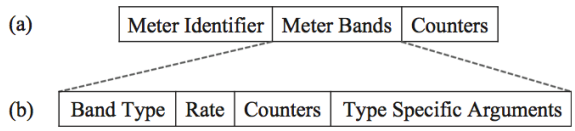
Figure 4-4, An overview of SDN architecture.

- Application Plane: An SDN application plane consists of one or more network applications (e.g. security, visualization, management etc.) that interact with controller(s) to utilize abstract views of the network for their internal decision-making processes. These applications communicate with controller(s) via an open Application-Controller Plane Interface, A-CPI (e.g. REST API).

An example of SDN forwarding device (Data Plane) is the OpenFlow switch. It is based on pipelines of flow tables defining how packets should be handled; each entry of a flow table consists of a matching rule, an action to be executed on matching packets (dropping, forwarding, etc.) and counters that maintain statistics of matching packets. These rules and actions of flow tables are installed by the SDN controller. Consequently, an OpenFlow switch can be programmed by the SDN controller to behave like a router, a switch, a firewall, or perform other roles (e.g., load balancing or traffic shaping) (Rao 2014) (Kreutz et al. 2015). The operation of an OpenFlow SDN forwarding device can be explained as follows:

When a new packet arrives at an OpenFlow switch, a path through a sequence of flow tables is selected. The lookup procedure begins in the first table and finishes either with a match or a miss (the latter occurring when no rule is found for that packet). A flow rule can be defined also by combining various matching fields. If there is no default rule, the packet will be discarded or forwarded to a non OpenFlow pipeline. Actions on matching packets can either be to forward the packet to an outgoing port, or encapsulate it and forward it to the controller, or to drop it, or to send it to the normal processing pipeline, or send it the next flow table. Data plane devices can be grouped into one or more separate controller domains. Table 4-1 illustrates the QoS-related features and changes implemented in the different versions of OpenFlow specification. Figure 4-6 shows, in diagrammatic form, an OpenFlow-enabled SDN controller with its OpenFlow switch.

Table 4-1, The QoS-related features and changes implemented in the different versions of OpenFlow specification.

OpenFlow Version	QoS features and changes implemented
OpenFlow 1.0 (Ben Pfaff, Brandon Heller 2009)	➢ An OpenFlow enabled switch can have one or more queues depending on its ports. ➢ There is an optional action called "enqueue[2]" which forwards packet through a queue attached to a port. ➢ An OpenFlow controller can query information about queues of a switch. However, the behaviour of the queue is determined outside the scope of OpenFlow, which can be configured through OF-CONFIG protocol (Open Networking Foundation (ONF) 2014a) but requires OpenFlow 1.2 or later versions. ➢ Also, header fields can include IP ToS, so packets can be matched against rules and their associated header fields.
OpenFlow 1.1 (Ben Pfaff, Bob Lantz 2011)	➢ Performs matching and tagging of VLAN and MPLS labels and traffic classes.
OpenFlow 1.2 (Open Networking Foundation (ONF) 2011)	➢ It has added an ability that enables a controller to query all queues in a switch. ➢ Another QoS related improvement in this version is that it has added a max-rate queue property. ➢ This version specifies that queues can be attached to ports and be used to map flows on them.
OpenFlow 1.3 (Open Networking Foundation (ONF) 2012)	➢ It introduces the rate limiting functionality by means of meter tables consisting of meter entries. ➢ A meter entry includes Meter Identifier, Meter Bands and Counters. A Meter Band consists of "Band Type" (e.g. drop or remark DSCP etc.), "Rate" (e.g. kb/s burst), "Counters" and optional "Type specific arguments", such as drop and DSCP remark, as seen in Figure 4-5. Counters may be maintained per-queue, per-meter, and per-meter band etc. They help controller collect statistics about the network. There may be one or more meter bands per meter table entry. ➢ Meters can be combined with the optional set queue action, which associates a packet to a per-port queue to implement complex QoS frameworks such as DiffServ. ➢ These meters allow for the rate-monitoring of traffic prior to output. More specifically, with meters, we can monitor the ingress rate of traffic as defined by a flow rule. Packets can be directed to a specific meter using the optional meter (meter_ id) instruction, where the meter can then perform some operations based on the rate it receives packets.  *Figure 4-5, OpenFlow 1.2-meter table.*
OpenFlow 1.4 (Open Networking Foundation (ONF) 2013)	➢ It presents the flow monitoring framework that allows a controller to monitor the changes done by other controllers to any subsets of the flow tables in real time. A controller can define a number of monitors, each selecting a subset of the flow tables. Each monitor includes a table id and a match pattern that defines the subset monitored. When any flow entry is added, modified or removed in one of the subsets defined by a flow monitor, an event is sent to the controller to inform it about the change.
OpenFlow 1.5 (Open Networking Foundation (ONF) 2014)	➢ It replaces the meter instruction, which was used for metering in previous versions (1.3 and 1.4), with a meter action. As a result, multiple meters can be attached to a flow entry, and meters can be used in group buckets.

[2] This action has been renamed to "set_queue" in OpenFlow 1.1 and later versions.

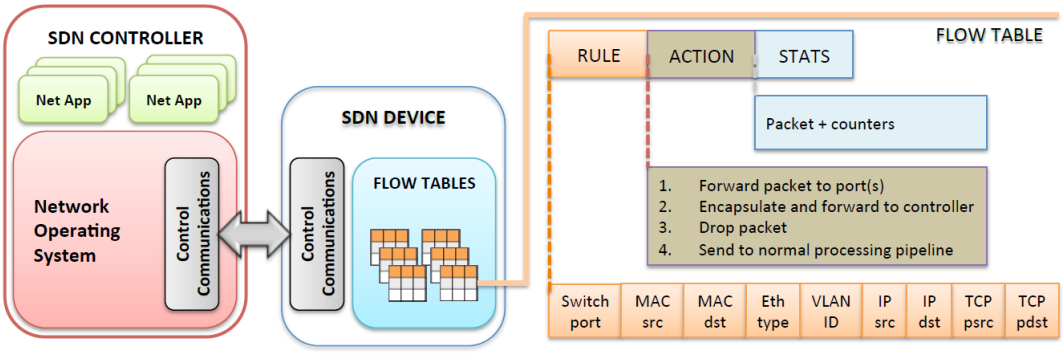


Figure 4-6, OpenFlow enable SDN controller and forwarding devices diagram.

The abovementioned specifications of OpenFlow switch, Table 4-1, does not currently provide support for queue configuration. The queue configuration in OpenFlow switch is handled by specific OpenFlow Management and Configuration Protocol, OF-CONFIG (Open Networking Foundation (ONF) 2014a) which is being standardized by ONF and Open vSwitch Database Management Protocol, OVSDB (IETF 2013) (Palma et al. 2014) which is standardized by the IETF.

OpenFlow switches can be grouped into one or more separate controller domains. The controller domain takes requests from the control application (management application) via an A-CPI (e.g. REST API) and accomplishes consolidated management and monitoring of OpenFlow switches via OpenFlow and OF-CONFIG protocols. There are two control models, which can be used in an SDN (Rao 2014) (Kreutz et al. 2015). These are the centralized controller model, whereby a single entity manages all OpenFlow switches; and the distributed controller model, which controls a cluster of nodes and controllers that are connected to each other using I-CPI or eastbound and westbound (API) interfaces to import/export data; such (API) interfaces are used to identify controllers and to provide compatibility among different kinds of controllers. The distributed controller model is more scalable and dependable than the centralized model. Figure 4-7 shows the connection of distributed controllers.

There is no standardized controller to provide queues management yet. There are many various SDN controller platforms offering different features for users. Table 4-2 summarizes some of the SDN controller projects with regards to their QoS support.

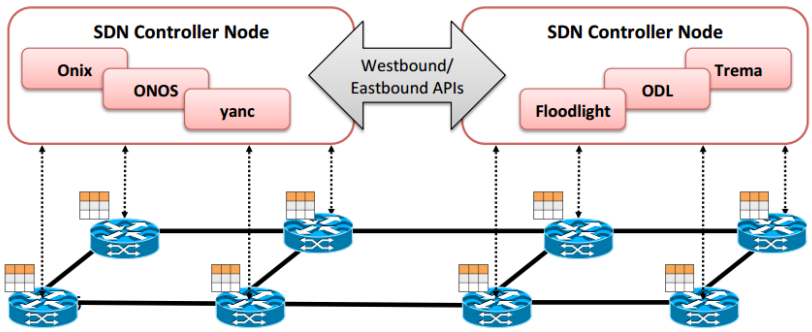


Figure 4-7, Distributed controllers' connections.

Table 4-2, Some of SDN controller projects with regards to their QoS support.

SDN Controller	Features
OpenDaylight (OpenDaylight Project 2013)	➢ It is an open-source controller platform. ➢ Programming language is Linux. ➢ It consists of many other sub-projects, such as C-DPI plugins (e.g. OpenFlow, NetCONF, SNMP, and BGP) and applications (e.g. DDoS Protection and Virtualization Coordinator), complementing each other to compose a complete reference controller platform for heterogeneous networks.
Open Network Operating System (ONOS Project 2015)	➢ It is a distributed SDN control platform aimed at improving scalability, performance and availability of networks for service providers. ➢ It is also an open-source platform. ➢ ONOS has limited QoS support currently. It supports OpenFlow metering mechanism, but this feature is rarely implemented in existing switches. The idea behind this support is based on implementation of OpenFlow "set_queue" functionality in ONOS.
Floodlight Project (Floodlight Project 2013)	➢ It is a Java-based open-source SDN controller. ➢ QoS module (Wallner & Cannistra 2013) implemented for Floodlight controller aims at providing an application that does matching, classification, flow insertion, flow deletion, and policy handling for QoS. The module utilizes OpenFlow 1.0 "enqueue" action and the network ToS bits. It controls tracking and storing services with their DSCP values, applying policies for services class, and tracking of policies in switches. ➢ Another QoS module implemented for Floodlight controller, the "QueuePusher" (Palma et al. 2014). It utilizes OVSDB protocol integrated with A-CPI API of Floodlight to generate appropriate queue configuration messages. The "QueuePusher" module uses a CRUD (Create, Read, Update, Delete) API, exposed by Floodlight, that allows external entities to manage Open vSwitch.

4.4 The Algorithm's Mathematical Model:

In this section, mathematical expressions for managing resources within the edge routers of a DiffServ domain and across different DiffServ domains at an update time interval *k* are developed. Figure 4-8 presents a representation of a service class queue at an update time interval *k*. According to queuing theory, the queuing system is considered in this model non-preemptive priority queuing (J. Virtamo 2005) such that:

- Each priority class has a separate logical queue,
- Queues are serving sequentially,
- When the scheduler/server becomes free, a packet from the head of the highest priority non-empty queue enters the scheduler,
- The packets interarrival times in the highest priority service class has a constant distribution while the packets interarrival times of other queues have expositional distributions.

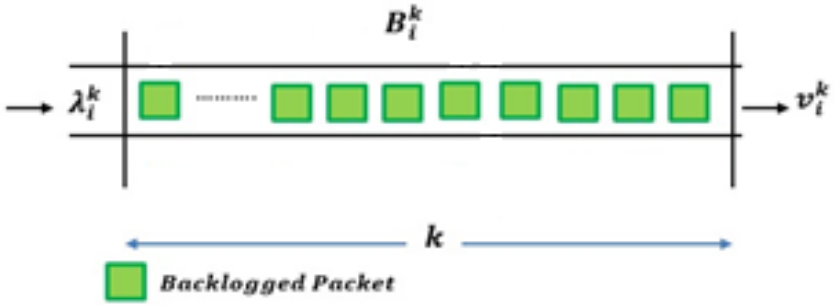


Figure 4-8, A representation of the service class queue at an update time interval K.

The proportional delay differentiation model (PDD) is a relative differentiation model. It states that "for all pairs of classes and for all time intervals $(t, t+\tau)$ in which both $\bar{d}_i(t, t+\tau)$ and $\bar{d}_j(t, t+\tau)$ are defined for service classes *i* and *j* respectively, the delay ratios between classes will remain constant and controllable independently of the class load (Chin-Chang Li et al. 2000), as shown in Equation 2-2.

The instantaneous value of the queue length q_i^k in bytes for a service class (*i*) at the update time interval *k* can be expressed in terms of the amount of class (*i*) backlogged packets $B_{i,}^k$ as shown in Equation 4-1:

$$q_i^k = B_i^k * L_i^k \quad \text{(Bytes)}$$

4-1

where:

B_i^k is the number of backlogged packets in the service class (*i*) at the update time interval *k*.

L_i^k is the average packet size in bytes for service class (*i*) at update time interval *k*.

The average service class queue length at update time interval *k* can be measured using an exponential weighted moving formula (Sharma et al. 2014) (Wang et al. 2001) as illustrated in Equation 4-2:

$$\overline{q_i^k} = (1 - \Delta) * \overline{q_i^{k-1}} + \Delta * q_i^k$$

4-2

where $\overline{q_i^k}$ is the average queue size at the update time interval *k*, q_i^k is the instantaneous queue size at the update time interval *k*, $\overline{q_i^{k-1}}$ is the average queue size at the previous update time interval *k-1* and Δ is a scaling factor. As illustrated in the next chapter, a simulation is used to choose a value of Δ such that the fluctuation in the instantaneous queue length is reduced.

From Figure 4-8, the packet (N) in the service class (*i*) queue suffers a delay (D_N) equal to the aggregate delays caused by the other packets that are awaiting service. This delay can be measured by taking the difference between the packet's dequeue time and its enqueue time, as demonstrated in Equation 4-3:

$$D_N = t_{Deq.}^N - t_{Enq.}^N$$

4-3

where D_N is the delay of packet (N) which has a dequeue time of $t_{Deq.}^N$ and an enqueue time of $t_{Enq.}^N$.

The average queue delay d_i^k for the service class (i) at the update time interval *k* represents the ratio of the aggregate cumulative delays of the backlogged packets to the total number of backlogged packets in the queue (*i*), as shown in Equation 4-4:

$$d_i^k = \frac{\sum_{j=1}^{B} D_j}{B}$$

4-4

where *B* represents the total number of backlogged packets in the service class (*i*) queue.

The scheduling rate v_i^k of service class (i) at an update time interval k represents the ratio of the average service class queue length to the average service class delay, as shown in Equation 4-5:

$$v_i^k = \frac{\overline{q_i^k}}{d_i^k}$$

4-5

From Equation 2-2, by substituting Equation 4-5 with respect to service classes (i) and (j) respectively we get Equation 4-6:

$$\frac{v_i^k}{v_j^k} = \frac{\frac{1}{\delta_i}\overline{q_i^k}}{\frac{1}{\delta_j}\overline{q_j^k}}$$

4-6

From Equation 4-6, it can be seen that the service class scheduling rate can be changed dynamically enabling it to maintain the delay differentiation parameter ratios between classes relative to the predefined specifications of the delay differentiation parameter for each class (δ) and the amount of backlogged packets in the service class queues. If there are three service classes (SC_1, SC_2 and SC_3) in a DiffServ domain and their delay differentiation parameters are ($\delta_1, \delta_2, \delta_3$) respectively, such that the SC_1 is considered the highest priority service class, SC_2 is considered the medium priority service class and SC_3 is considered the lowest priority service class. The scheduling rates for these service classes ($v_{SC1}^k, v_{SC2}^k \; and \; v_{SC3}^k$) at an update time interval k can be calculated as shown in Equation 4-7 and Equation 4-8. For the highest priority service class SC_1; which occupy the required bandwidth for its traffic, its scheduling rate at an update time interval k can be calculated as below:

$$v_{SC1}^k = \frac{\overline{q_{SC1}^k}}{d_{SC1}^k}$$

4-7

For the medium and lowest priorities service classes SC_2, SC_3 respectively, using the relation between service class scheduling rates between any two service classes, Equation 4-6, the scheduling rates for these classes at an update time interval k:

$$v_{SC2}^k = \frac{\delta_1}{\delta_2} * v_{SC1}^k * \frac{\overline{q_{SC2}^k}}{q_{SC1}^k} \text{ and } \qquad v_{SC3}^k = \frac{\delta_1}{\delta_3} * v_{SC1}^k * \frac{\overline{q_{SC3}^k}}{q_{SC1}^k}$$

4-8

Where:

- (δ_1, δ_2 and δ_3): Delay differentiation parameters for service classes (SC_1, SC_2 and SC_3) respectively, which are configurable parameters set by the DiffServ domain administrator.
- ($\overline{q_{SC1}^k}$, $\overline{q_{SC2}^k}$ and $\overline{q_{SC3}^k}$): Average queue size for service classes (SC_1, SC_2 and SC_3) respectively at an update time interval *k*. It can be calculated using Equation 4-2.
- d_{SC1}^k: Average SC1 queue delay at an update time interval k. It can be calculated using Equation 4-4.

For the multi core router DiffServ domains in scenarios 2 and 3 in sections 6.3.2 and 6.3.3 respectively, each out-port of a DiffServ router contains a number of service class queues. The number of these queues is equivalent to the number of defined service classes within that DiffServ domain. All the service class queues at each out-port of all the ingress routers of a multi core DiffServ domain have the same scheduling rate configuration. This rate can be set by selecting a port with the maximum average service class queue length and a port with the minimum average SC_1 queue delay because the service class scheduling rate is directly proportional to the average service class queue length and inversely proportional to the average service class queue delay. These procedures lead to the maximization of throughput for the highest priority service class and the other classes within a multi-core DiffServ domain.

To illustrate the procedure of calculating the maximum average queue size for a service class *i* ($\overline{(Q_{SCi}^k)}$) for a router consists from *j* ports at an update time interval *k*; the average queue size for a service class *i* ($\overline{q_{SCi}^k}$) at each port of a multi-core router at an update time interval *k* is calculated first using Equation 4-2 then the maximum average queue size for this class ($\overline{(Q_{SCi}^k)}$) at all ports of a multi-core router at an update time interval k can be calculated as below, Equation 4-9:

$$\overline{Q_{SCi}^k} = max.\left(\overline{q_{SCi1}^k}, \overline{q_{SCi2,}^k} \dots\dots, \overline{q_{SCij}^k} \right) \qquad 4\text{-}9$$

where *j* represents the number of ports of a multi-core router.

The same procedure is applied when calculating the minimum average SC_1 queue delay (D_{SC1}^k) at all ports of a multi-core router and at an update time interval k, Equation 4-10:

$$D_{SC1}^{k} = min.\left(\overline{d_{SC11}^{k}}, \overline{d_{SC12,}^{k}} \ldots\ldots, \overline{d_{SC1j}^{k}} \right)$$

4-10

Where *j* represents the number of ports of a multi-core router.

The service classes scheduling rates for the multi-out-port ingress routers (or multi-core router) at an update time interval *k* with considering the maximum average service class queue size and minimum average SC1 queue delay can be illustrated as shown in Equation 4-11 and Equation 4-12:

For the highest priority service class SC1, considering the maximum average SC1 queue size and minimum average SC1 queue delay, Equation 4-11:

$$v_{SC1}^{k} = \frac{\overline{Q_{SC1}^{k}}}{D_{SC1}^{k}}$$

4-11

For the medium and lowest priorities service classes SC2, SC3 respectively, using the relation between service class scheduling rates between any two service classes, Equation 4-6 and the maximum average service class queue size. The scheduling rates for these classes at an update time interval *k*:

$$v_{SC2}^{k} = \frac{\delta_1}{\delta_2} * v_{SC1}^{k} * \frac{\overline{Q_{SC2}^{k}}}{Q_{SC1}^{k}} \text{ and } v_{SC3}^{k} = \frac{\delta_1}{\delta_3} * v_{SC1}^{k} * \frac{\overline{Q_{SC3}^{k}}}{Q_{SC1}^{k}}$$

4-12

where:

- $\overline{Q_{SC1}^{k}}$ is the maximum average service class (SC_1) queue length at any port of the multi-port ingress routers at an update time interval *k*.
- D_{SC1}^{k} is the minimum average service class (SC_1) queue delay at any port of the multi-port ingress routers at an update time interval *k*.
- $\overline{Q_{SC2}^{k}}$ and $\overline{Q_{SC3}^{k}}$ are the maximum average service class (SC_2 and SC_3) queue lengths respectively at any port of the multi-port ingress routers at an update time interval *k*.

The weights used by the WFQ scheduling algorithm can be changed dynamically within an update time interval *k* by changing the scheduling rates of the service classes according to Equation 4-8. The weight of a service class represents its share of the router's out link

bandwidth (BW). The weight can be determined by calculating the Greatest Common Divisor (GCD) of the normalised percentage values of the rates of all the service classes.

The normalised percentage value $\overline{\mathrm{SC}_i^k}$ of a service class' (i) scheduling rate can be calculated as shown in Equation 4-13:

$$\overline{\mathrm{SC}_i^k} = \frac{\mathrm{v}_{sci}^k}{\sum_{j=1}^{M} \mathrm{v}_{scj}^k} * 100\%$$

4-13

The weight of service class i (W_i^k) within the update time interval k can be calculated as shown in Equation 4-14:

$$\left.\begin{array}{c} \mathrm{W}_i^k = \dfrac{\overline{\mathrm{SC}_i^k} * \text{Output link capacity (bits)}}{100 * \mathrm{X} * \text{Average packet size in queue (bits)}} \\ \mathrm{X} = \mathrm{GCD}\left(\overline{\mathrm{SC}_1^k}, \overline{\mathrm{SC}_2^k}, \ldots\ldots, \overline{\mathrm{SC}_M^k}\right) \end{array}\right\}$$

4-14

where M is the number of service class queues in the DiffServ domain.

All the core routers within the DiffServ domains forward the service class packets according to the fixed weights. These weights are configured by the domain administrator and chosen such that the priority levels of the service classes are maintained.

In order to manage resources across different DiffServ domains in cases where any service class queue at the ingress router of the downstream domain suffers from traffic congestion, the congestion level (α_i^{k+1}) for the service class queue i within an update time interval (k+1) is calculated as shown in Equation 4-15:

$$\alpha_i^{k+1} = \frac{\overline{\mathrm{q}_i^k}}{\text{Buffer Size}}$$

4-15

Each service class queue has threshold parameters ($\mathrm{T}_{\mathrm{i(High)}}$ and $\mathrm{T}_{\mathrm{i(Low)}}$). These two parameters define the lower and higher boundaries of congestion levels in the service class queue i.

The lower boundary ($\alpha_{i(Low)}$) is a configurable parameter, it represents the ratio of the Low congestion threshold parameter ($\mathrm{T}_{\mathrm{i(Low)}}$) for the service class queue i to the service class buffer size as shown in Equation 4-16:

$$\alpha_{i(Low)} = \frac{\mathrm{T_{i(Low)}}}{\text{Buffer Size}}$$

4-16

The higher boundary ($\alpha_{i(High)}$) is also a configurable parameter, it represents the ratio of the Higher congestion threshold parameter ($\mathrm{T_{i(High)}}$) for the service class queue *i* to the service class buffer size as shown in Equation 4-17:

$$\alpha_{i(High)} = \frac{\mathrm{T_{i(High)}}}{\text{Buffer Size}}$$

4-17

The measured congestion level α_i^{k+1} is compared with the two threshold parameters $\alpha_{i(High)}$ and $\alpha_{i(Low)}$ of that service class queue. If the congestion level is below $\alpha_{i(Low)}$ then there is no need to reduce the scheduling rate for the equivalent service class queue at the egress router of the upstream DiffServ domain. If α_i^{k+1} is within the congestion threshold range $\alpha_{i(Low)}$ and $\alpha_{i(High)}$, then the scheduling rate $\left(\mathrm{v}_i^{k+1}\right)$ for the equivalent service class queue *i* at the egress router of the upstream DiffServ domain is reduced by a congestion rate update factor (β_i^{k+1}) at the next update time interval (*k+1*). For multi-port ingress routers, the measured congestion level (α_i^{k+1}) is set to that of the port with the maximum average service class queue length, i.e.: assume a multi-core router consists from *j* ports, the congestion level (α_i^{k+1}) for a service class queue at each port is calculated first using Equation 4-15 at an update time interval k then the maximum congestion level ($\propto_{\mathrm{i(max.)}}^{\mathrm{k+1}}$) for this service class at all ports and at an update time interval k as shown in Equation 4-18.

$$\propto_{i(max.)}^{k+1} = \max(\propto_{i1}^{k+1}, \propto_{i2}^{k+1}, \dots\dots, \propto_{ij}^{k+1})$$

4-18

where *j* represents the number of ports of a multi-core router.

The scheduling rate update factor (β_i^{k+1}) for the service class queue *i* within an update time interval (*k+1*) is chosen such that it is linearly proportional to the measured congestion level (α_i^{k+1}). This means that the congestion update factor (β_i^{k+1}) increases with increments in the congestion level (α_i^{k+1}) when $\alpha_{i(Low)} < \alpha_i^{k+1} < \alpha_{i(High)}$. Figure 4-9 illustrates the relationship between the congestion level $\left(\alpha_i^{k+1}\right)$ and the scheduling rate update factor (β_i^{k+1}).

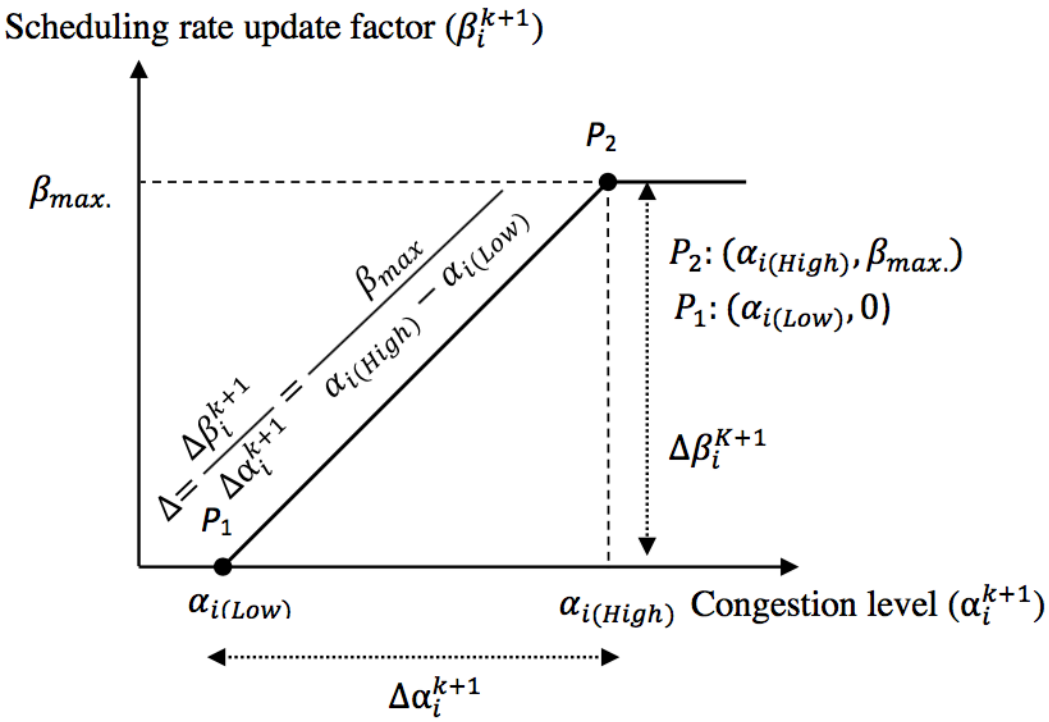


Figure 4-9, Relation between the congestion level and the scheduling rate update factor.

In order to develop an expression for the scheduling rate update factor (β_i^{k+1}) for $\alpha_{i(Low)} < \alpha_i^{k+1} < \alpha_{i(High)}$, the line equation above is taken to represent an expression for this factor (β_i^{k+1}). This expression can be formulated as below:

$$y = mx + b$$

$$\beta_i^{k+1} = m\alpha_i^{k+1} + b$$

4-19

Where y represents the scheduling rate update factor (β_i^{k+1}) for the service class queue i within an update time interval ($k+1$), x represents the congestion level, (α_i^{k+1}) for the service class queue i within an update time interval ($k+1$), m is the slope of the line and b is a constant which represent the line's intersection point with the y axis. The slope of this line m can be expressed as illustrated in Equation 4-20:

$$m = \frac{\Delta y}{\Delta x} = \frac{\Delta\beta_i^{k+1}}{\Delta\alpha_i^{k+1}} = \frac{\beta_{max.}}{\alpha_{i(High)} - \alpha_{i(Low)}}$$

4-20

By substituting either P_1 or P_2 of Figure 4-9 and the value of slope m in Equation 4-19, the scheduling rate update factor (β_i^{k+1}) for $\alpha_{i(Low)} < \alpha_i^{k+1} < \alpha_{i(High)}$ can be expressed as shown in Equation 4-21:

$$\beta_i^{k+1} = \beta_{max.} \frac{\alpha_i^{k+1} - \alpha_{i(Low)}}{\alpha_{i(High)} - \alpha_{i(Low)}} \qquad 4\text{-}21$$

If the congestion level exceeds the α_{High}, then the scheduling rate update factor (β_i^{k+1}) is set to a fixed configurable value ($\beta_{max.}$).

where, ($T_{i(Low)}$) and ($T_{i(High)}$) are the low and high threshold parameters for the service class queue as configured by the DiffServ domain administrator. v_i^k represents the scheduling rate for the service class queue at the egress router of the upstream domain according to its traffic state at the current update interval *k*. v_i^{k+1} represents the scheduling rate for the service class queue at the egress router of the upstream domain according to the traffic states in the upstream and downstream DiffServ domains at the next update interval *k+1*.

By reducing the scheduling rate of the congested service class at the egress router of the upstream domain, the service class queue's normalised values and weights are also changed. Consequently, the bandwidth resource of the link that connects the DiffServ domains is redistributed in a different way such that it may be possible to improve the performance of some service classes in the case of traffic congestion in a downstream DiffServ domain.

4.5 Chapter Summary:

This chapter presents the DRAM-NFV algorithm for managing resources within and across different DiffServ domains. An overview of the framework that can be used to enable the use of this algorithm has been provided. In addition, all the mathematical expressions that are related to the resources management within Proportional Delay DiffServ routers and across different administration Proportional Delay DiffServ domains have been introduced in this chapter. These mathematical expressions have been used to create a DiffServ domain module and implement the algorithm test scenarios using a simulation environment, as we will see in the next two chapters.

Chapter Five
DRAM-NFV Algorithm Implementation using a Simulated Environment

5.1 Introduction:

The procedures related to the implementation of the DiffServ module and to the application of the DRAM-NFV algorithm within a simulated environment are described in this chapter. The network simulation platform NS3 has been used for this research. Three main application programs are built from scratch using the NS3 module library and the C++ language. These include the network queuing program which represents the differentiation queuing model for the proportional delay DiffServ domains, the application traffic program which generates different types of network traffic and the network topology program that represents the topologies of the test scenarios. Code listings of the algorithm implementation is found in Appendix A-3. All these application programs work together in order to facilitate an environment in which a prototype of the (DRAM-NFV) algorithm could be constructed. Application traffic will be discussed in the next chapter while this chapter covers the DiffServ network queuing simulation model.

5.2 The (NS3) Network Simulation Program:

The Network Simulator (NS3) is one of a number of discrete event network simulators available. The term discrete event network simulator means a framework within which the process of codifying the behaviours of a complex system as a set of well-defined events takes place which allows for their simulation by the execution of these events in a scheduled manner. Conceptually, the simulator keeps track of a number of events that are scheduled to execute at a time prescribed by other events. The job of the simulator is to execute the events in a logical order. Once the completion of an event occurs, the simulator will move to the next event (or will exit if there are no more events to be processed).

There are many available discrete events simulators such as Opnet, Matlab, NS2, NS3, and etc. The NS3 simulator is an open source piece of software, licensed under the GNU GPLv2 license. It is available publicly for research and dedicated to research and educational purposes. The NS3 simulation software supports research on both IP and non-IP based networks. It should be noted that NS3 is different from NS2, and NS3 does not support NS2 models. NS2 (NS-2 lists

contributors. 1995) is implemented using a combination of oTCL (for scripts describing the network topology) and C++ (The core of the simulator), making the debugging process complex, while NS3 is written in C++ and Python programming languages. These latter programming languages can be optionally used as an interface, making the debugging process easier than it is for NS2; a knowledge of just C++ is sufficient, in terms of languages. The NS3 simulator version (3.23) is used to simulate the test scenarios that will be described in the next chapter; this is the current version of NS3, released in May 2015. There are some educational materials available for those who wish to learn about NS3 and the C++ programming language. These are (NS3 Tutorial 2015), (NS3 Manual 2015), (NS3 Model library 2015),(NS3 Doxygen 2015),(NS3 software users group) and (C++ Programming Language 2016).

5.3 Queue Model Simulation:

One of the challenges of NS3 is that it does not cover all the functions needed to represent the functionalities of a DiffServ router as required for this research. These functionalities include:

I. sharing multiple service class queues to one out link;
II. packets classification, dropping and scheduling mechanisms for service class queues; and
III. DiffServ queue and link performance measurements in terms of average queue delays, average queue lengths, average queue scheduling rates, DiffServ link utilization and average end-to-end delay metrics for service classes traffic.

Building multiple queues using standard (NS3::Queue Class Reference) is required multiple output link interfaces; this means that each queue needs an output link interface. This is not acceptable when implementing a DiffServ domain. In order to implement the test scenarios of sections 6.3.1, 6.3.2 and 6.6.3 using the NS3 simulation environment, a new queuing model had to be designed covering all of the abovementioned functions but using some of the classes defined in the NS3 simulator and the C++ programming language packages. This new queueing model is called “RDWQueue”, and it forms the main part of the specifically written simulation code. The (RDWQueue) model is inherited or derived from the NS3 queue class reference in order to provide a consistent way to enqueue and dequeue packet from service class sub queues or queues.

This new queueing module had to be defined within the modules of the NS3 package. The procedures for adding a new module to the NS3 package is illustrated in (NS3 project 2010). Appendix A-2.1 illustrates the NS3 class references which are used to build the RDWQueue model and their purposes.

The packets in RDWQueue will be banded into sub queues according to their service class tags. All these bands or sub queues[3] must share one out-link in order to forward their traffic. For each service class sub queue, there are configurable parameters which describe the behaviour of that service class sub queue, these parameters are as follows.

I. **Buffer Size** - the maximum capacity of the service class sub queue in bytes.
II. **Lower and Higher threshold queue length parameters** – these parameters specify the lower and higher boundaries of the service class queue length (in bytes) to the WRED packet queue management algorithm.
III. **Lower and Higher packet drop probabilities –** these two parameters specify the threshold values of packet drop probability to the WRED packet queue management algorithm.
IV. **Lower and Higher threshold of congestion levels –** these parameters specify the lower and higher boundary values for the service class congestion condition and are used in managing resources across different DiffServ domains.

To achieve a differentiation among these bands or sub queues in the RDWQueue model, a packet scheduling scheme (Scheduler) is also implemented in order to forward packets from these sub queues to the one out-link. This scheduler is based on calculating the normalized percentage values for the service classes scheduling rates as explained in the previous chapter. The weights for the implemented DiffServ scheduler can be configured dynamically in the case of the edges routers of a DiffServ domain or pre-configured as fixed weights in the case of the core routers of a DiffServ domain. The most important functions that the RDWQueue model can perform are:

5.3.1 Initialising the Simulation:

An object of the RDWQueue model must be instantiated when creating the queues for the test scenarios. The constructor function (RDWQueue::RDWQueue(int id)) is used for this purpose. Constructors are typically used also (as here) to initialize member variables of the class to appropriate default values; a destructor is executed when an object of a class is destroyed. In the constructor function, the periodic processes of calculating the scheduling rates for the service class queues, producing statistical reports about the queues, obtaining the out-link service classes traffic utilizations of the DiffServ router and the number of forwarded packets

[3] In this thesis, the terms sub-queue and queue are used frequently. Both have the same meaning: the service class queue.

from sub queues – all in relation to a standard interval *k* - are also defined and need to be identified and scheduled-in by the NS3 scheduler during the simulation run.

5.3.2 Traffic Queuing:

When an incoming packet arrives at the DiffServ router, it must be accommodated or queued in a suitable service class queue, compatible with its service type. The (RDWQueue::DoEnqueue (Ptr<Packet> p)) function performs the queuing process of a DiffServ router.

The Enqueue function takes the packet from the input link of the RDWQueue. It inspects the packet service class tag using the RDWQueue classification function (Classify) and inserts it into the appropriate sub queue if the service class (x) sub queue size (SCxByteLength) does not exceed the lower threshold queue length parameter (REDSCxTLOW) for that service class sub queue (x). After that, the sub queue state of the service class needs to be updated and the sub queue size (SCxByteLength) for service class (x) should be increased by the size of the inserted packet. In addition, the unique ID and the simulation time of the inserted packet must be stored into (SCxEnqueueTime) so that the packet delay can be calculated later. If the packet is unknown, then the Enqueue function insert it in any service class queue depending on the available resources in these queues.

If the service class (x) sub queue size exceeds the lower or higher threshold queue length parameters for that service class sub queue (REDSCxTLOW) or REDSCxTHIGH) respectively, then random numbers of service class packets will be dropped. Based on the WRED configuration of the Cisco™ [4] 10000 series QoS routers (Cisco 2013), the lower and higher drop probabilities (REDSCxLDROP or REDSCxHDROP) of each service class queue should not exceed on 0.1. An exponential random distribution function called (GenRand) is designed using (NS3::Exponential Random Distribution) to generate a random number (Ran) ranging between 0 and 0.5 to determine how many packets should be dropped from a service class queue. If (Ran) is outside the range defined by the Lower or Higher packet drop probability values (0.1), then the packet will be inserted into the service class sub queue, otherwise it will be dropped and the function will increase the counter that counts the number of dropped packets (SCxdropcount) for that service class (x) by one. In addition, this function also drops packet if the service class (x) sub queue size (SCxByteLength) exceeds the service class (x) buffer size (SCxBYTEBUFFER). Figure 5-1 illustrates the flow chart of the Enqueue function and the code of this function is shown in page 147 of Appendix A-3.

[4] Cisco™ is a trade mark of Cisco Systems, Inc. and its affiliates.

- There are three defined service class queues (SC_1 queue, SC_2 queue, SC_3 queue).
- x represents the QoS tag or band of a received packet. It could be SC_1 or SC_2 or SC_3 or unknown.

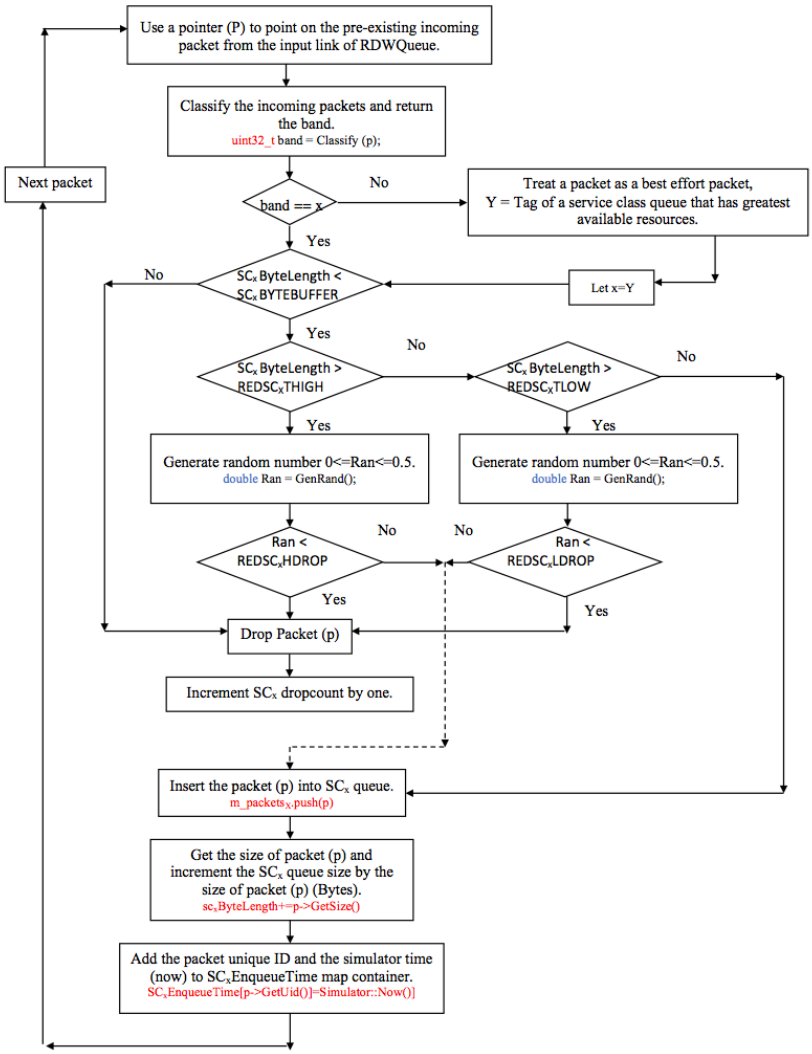


Figure 5-1, Flow chart illustrating the enqueuing process.

5.3.3 Traffic Scheduling:

A scheduler is needed for forwarding the packets from the service class queues to the out link of the DiffServ router based on the weights that are allocated for the service classes. The (RDWQueue::DoDequeue (void)) function performs the scheduling function of a DiffServ router. It forwards packets from the service class sub queues to the out link of a DiffServ router based on their weights. These weights are configured either dynamically for the edge routers of a DiffServ domain or pre-configured as fixed weights for the core routers of a DiffServ domain. The weights represent the number of packets that should be forwarded from the service class sub queues. The procedures for specifying the weights of the service class sub queues are explained later in this chapter. Where there are three service classes within a

DiffServ domain, this forwarding scheme keeps to the following sequence for packet forwarding (SC_1, SC_2, SC_3, SC_1, SC_2, SC_3, ….). After forwarding a packet from a service class sub queue, the sub queue state must be updated; the sub queue size (SCxByteLength) for service class (x) should be decreased by the size of the forwarded packet in bytes. Moreover, the unique ID and simulation time of the send packet event should be stored into (SCxDequeueTime) so that the packet delay can be calculated later. The DiffServ router out link utilization for service class (x) should also be increased by the size of the sent packet in bytes. This scheduler must always be active; if there is no packet in a service class sub queue then the scheduler should move to the next sub queue to forward its packets according to their weights. Figure 5-2 illustrates the flow chart of the Dequeue function and the code of this function is shown in page 156 of Appendix A-3.

5.3.4 Classification of Incoming Packets:

The incoming, marked packets should be classified before being inserted into their dedicated queues. The function (RDWQueue::Classify (ptr<const Packet> p) is used to classify incoming packets. The code of this function is shown in page 162 of Appendix A-3.

- *If the DiffServ domain supports three service classes (SC): SC_1, SC_2, and SC_3.*

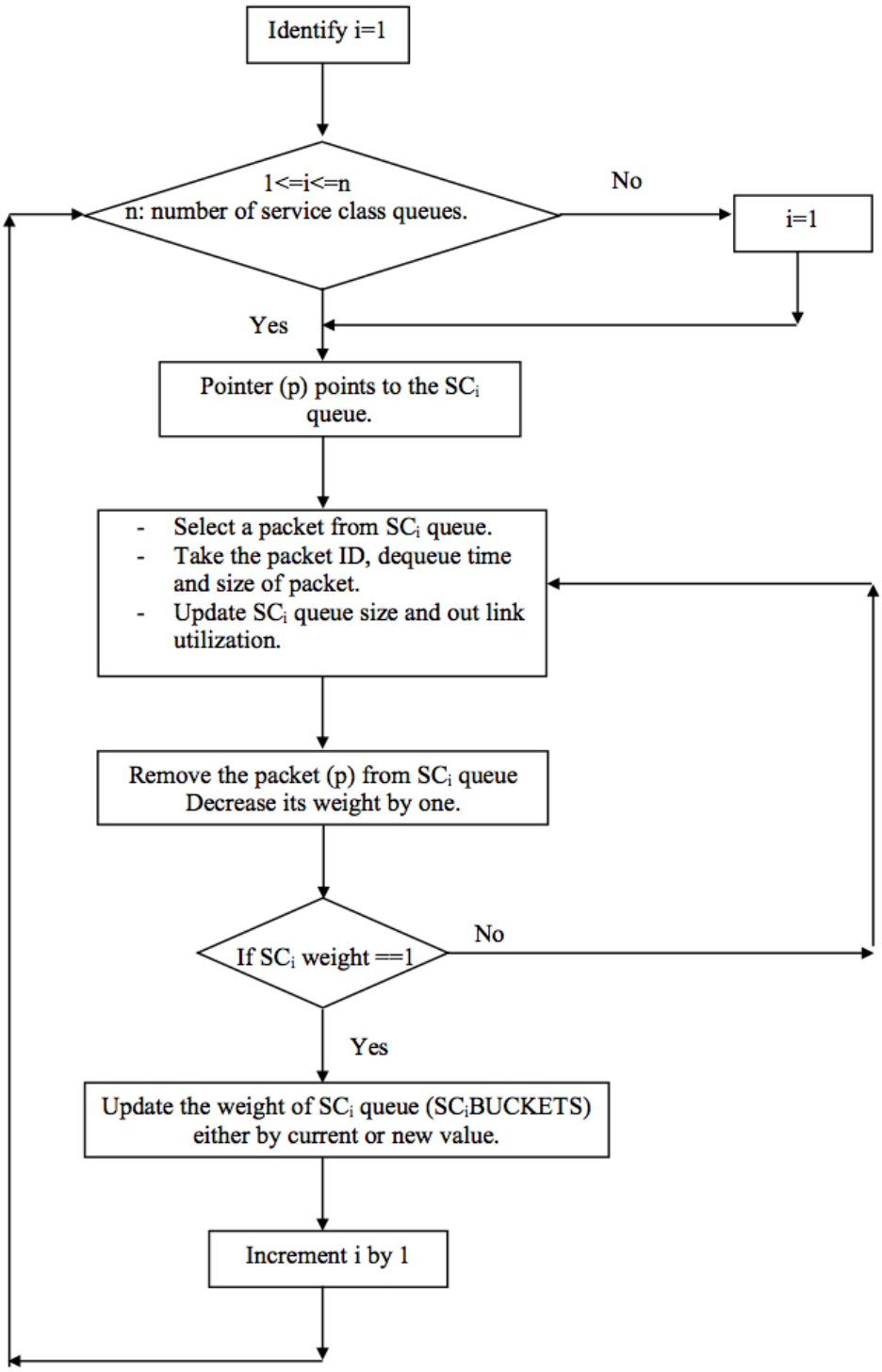


Figure 5-2, Flow chart illustrating the scheduling (dequeuing) process.

5.3.5 Calculation of Service Classes' Scheduling Rates:

The procedure for calculating service classes' scheduling rates must be performed by the edge routers of a DiffServ domain. The function RDWQueue::CalculateSchedulingRateByte performs this task. Within an interval of *k sec*, the function measures the average queue length, the average queue delay and the scheduling rate for each service class sub queue by using the mathematical equations Equation 4-2, Equation 4-3, Equation 4-4 and Equation 4-8 that were given in the previous chapter.

5.3.5.1 Measuring the Average Service Class Queue Length:

The average service class queue length can be calculated by using the mathematical equation Equation 4-2 which was explained in the previous chapter and as illustrated in the flow chart of Figure 5-3. The code of this process is shown in page 167 of Appendix A-3.

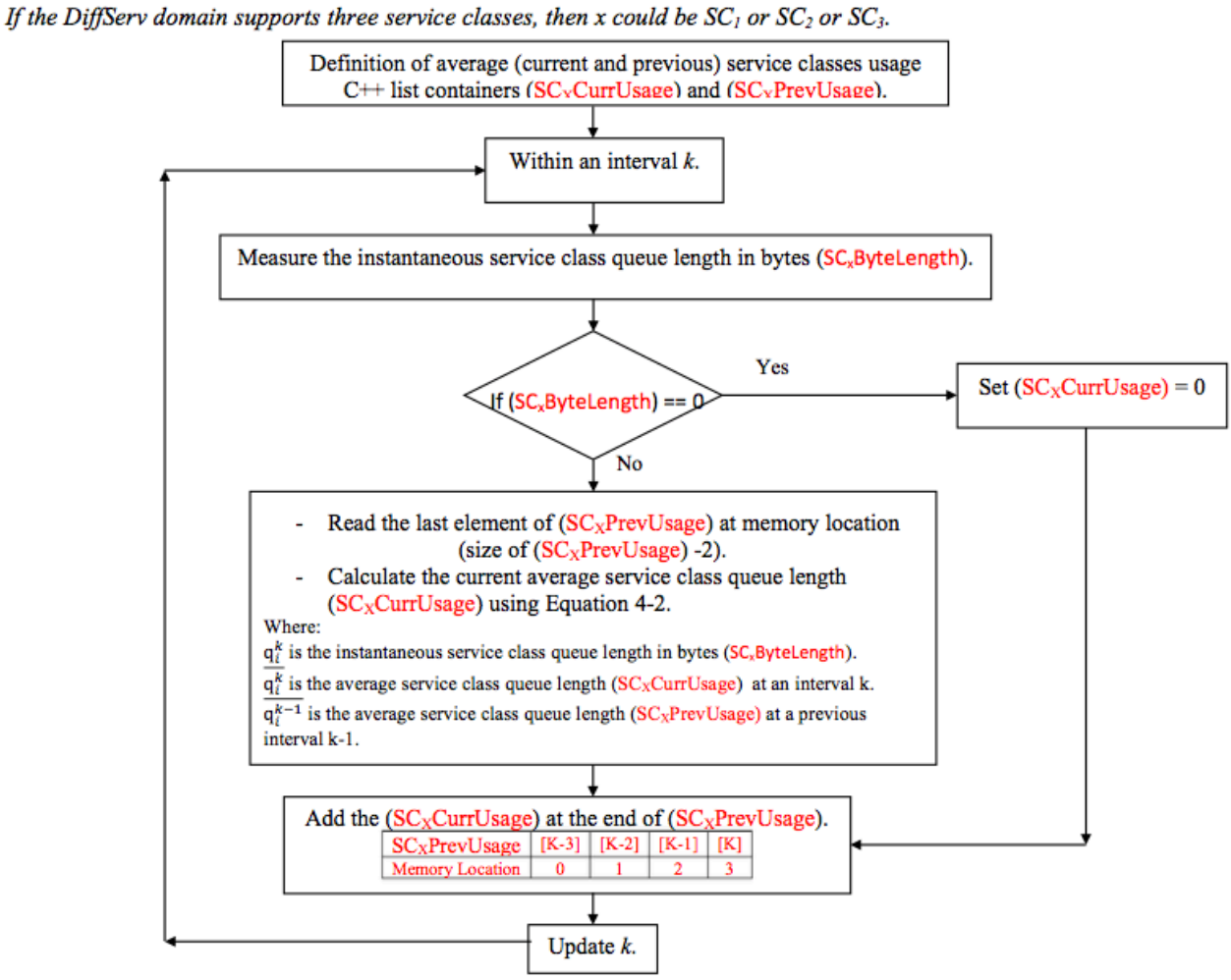


Figure 5-3, Flow chart illustrating the process of calculating the average queue length for service class queues.

5.3.5.2 Determining the Optimum Scaling Factor (Δ):

An experiment was conducted, using the NS3 simulator, to estimate an optimum value for the scaling factor Δ used within the estimated average service class queue length Equation 4-2. The value of Δ should be between 0 and 1. In the experiment, four values of Δ were tried (0.9,0.5, 0.1 and 0.01) in order to find an estimated average service class queue length from the instantaneous queue length value. The result of this experiment is illustrated in Figure 5-4. The dashed curve represents the instantaneous value of the service class queue length while the other curves represent the estimated average service class queue length at the above indicated values of Δ. When Δ is equal to 0.9 or 0.5, the estimated average queue length fluctuates and this will affect the differentiation pattern between service classes through the destabilizing of the scheduling rates for the service classes. When Δ equals 0.01, the estimated average queue length does not reflect the current queue length, although here the value does not fluctuate. When Δ is equal to 0.1, the fluctuation in the estimated average queue length is reduced and the result of the equation reflects an approximate view of the current service class queue length. Thus, Δ=0.1 is used in the simulation as an optimum value for the scaling factor Δ for the estimated average service class queue length Equation 4-2.

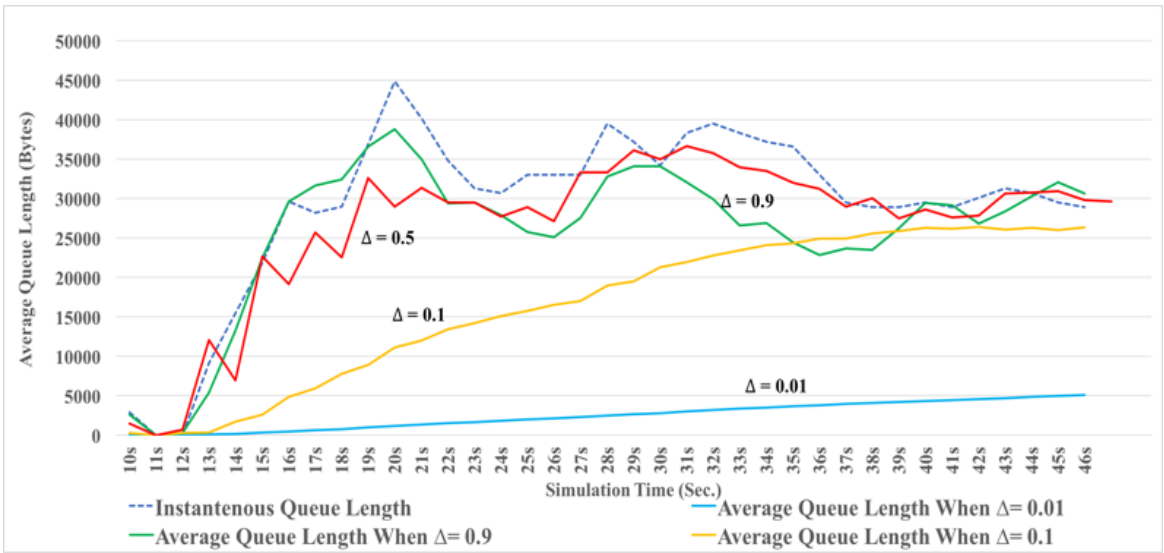


Figure 5-4, Estimating an optimum value for the scaling factor Δ.

5.3.5.3 Measuring the Average Service Class Queue Delay:

The packet delay represents the difference in time between a packet's insertion time into a queue and its departure time from the queue as explained in relation to Equation 4-3 of the previous chapter. Figure 5-5 illustrates the flow chart for calculating the packet delay of service class (x).

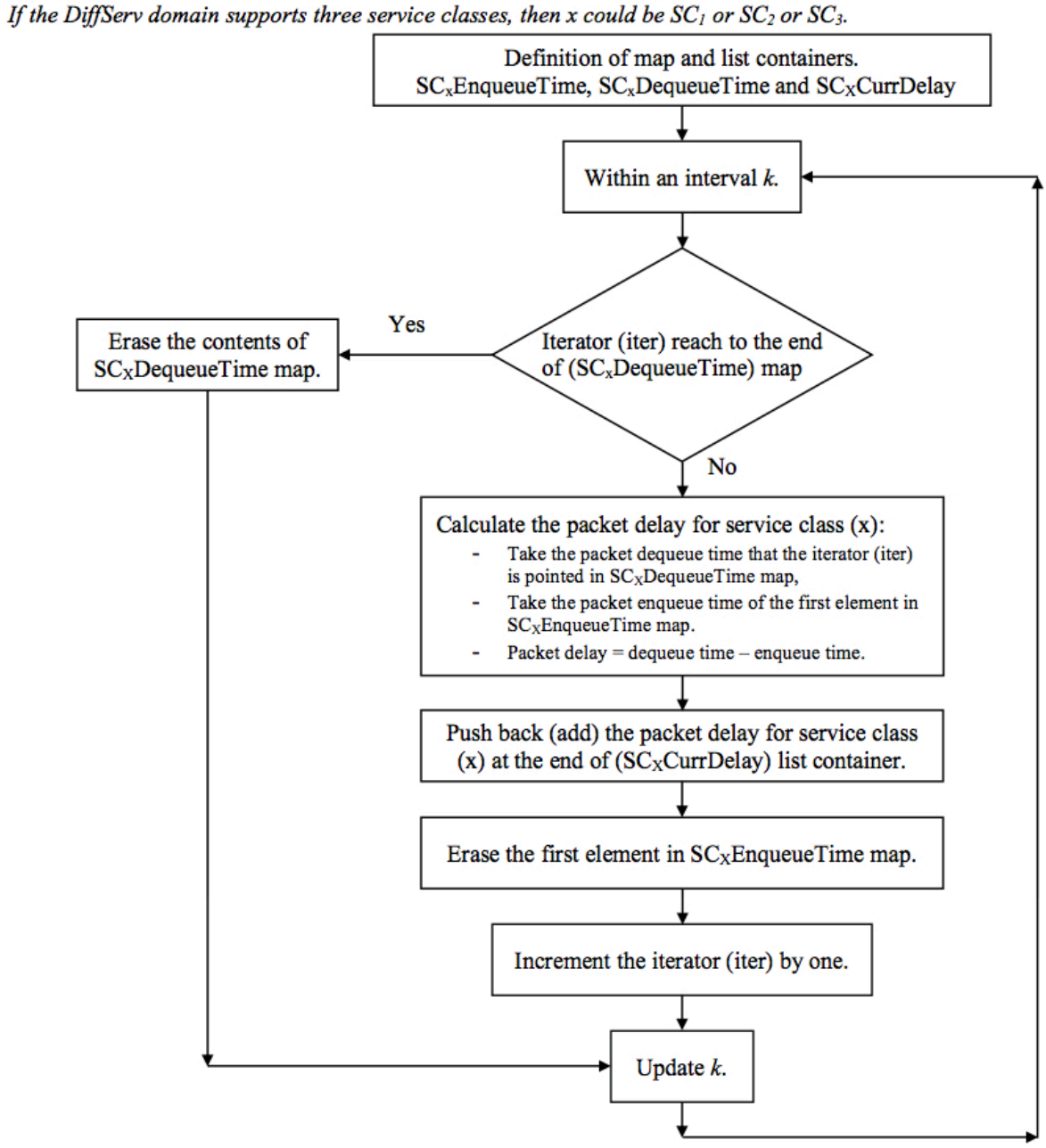


Figure 5-5, Flow chart illustrating the process of calculating the packet delay for a service class.

The average queue delay is equal to the aggregate of the packet delays in the queue divided by the number of packets in the queue, as explained in relation to Equation 4-4 of the previous chapter. Within an interval *k*, this delay can be calculated by taking the average of the packet delay values. Figure 5-6 illustrates the flow chart for calculating the average service class queue delay and the code of this process is shown in page 169 of appendix A-3.

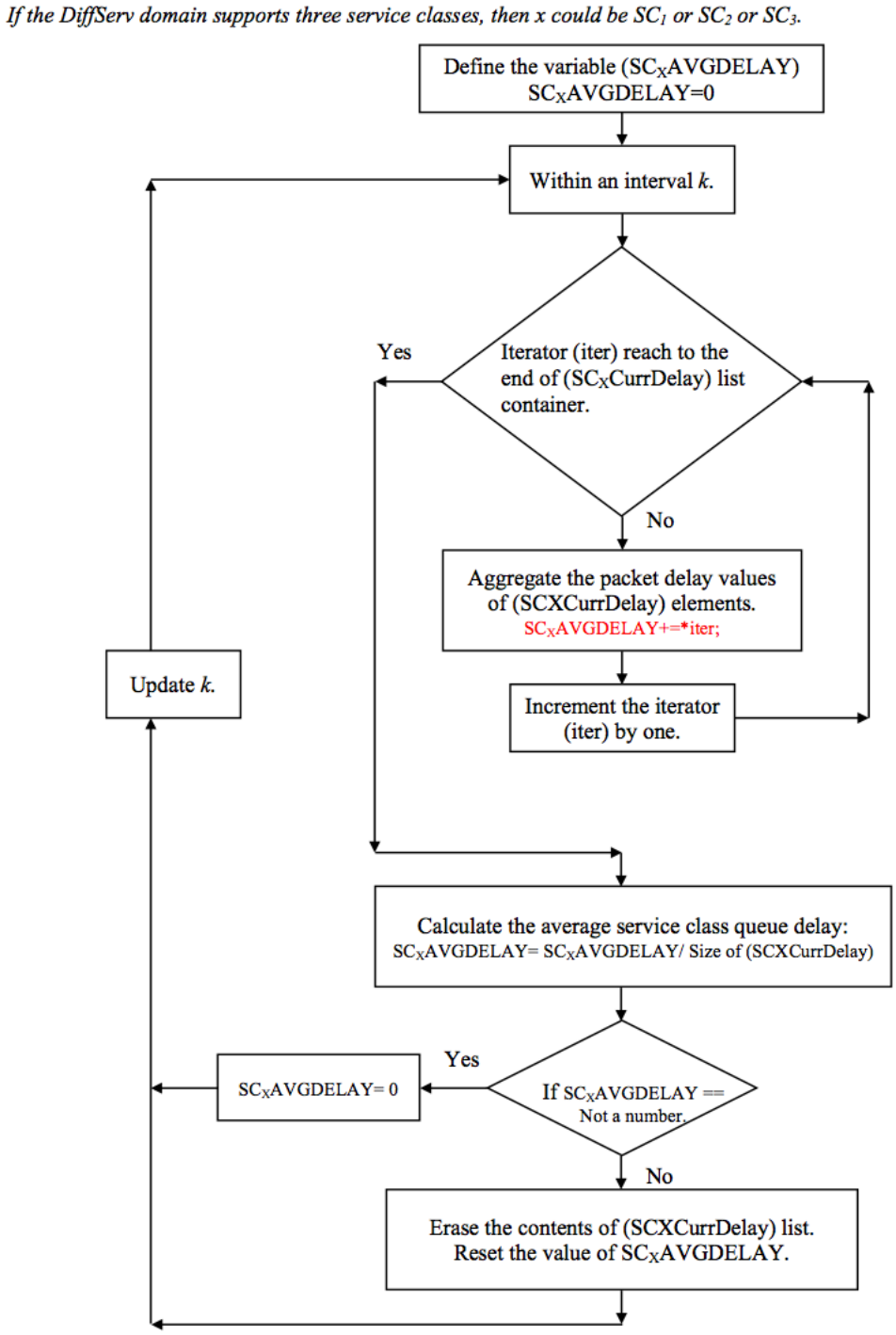


Figure 5-6, Flow chart illustrating the process of calculating the average service class queue delay.

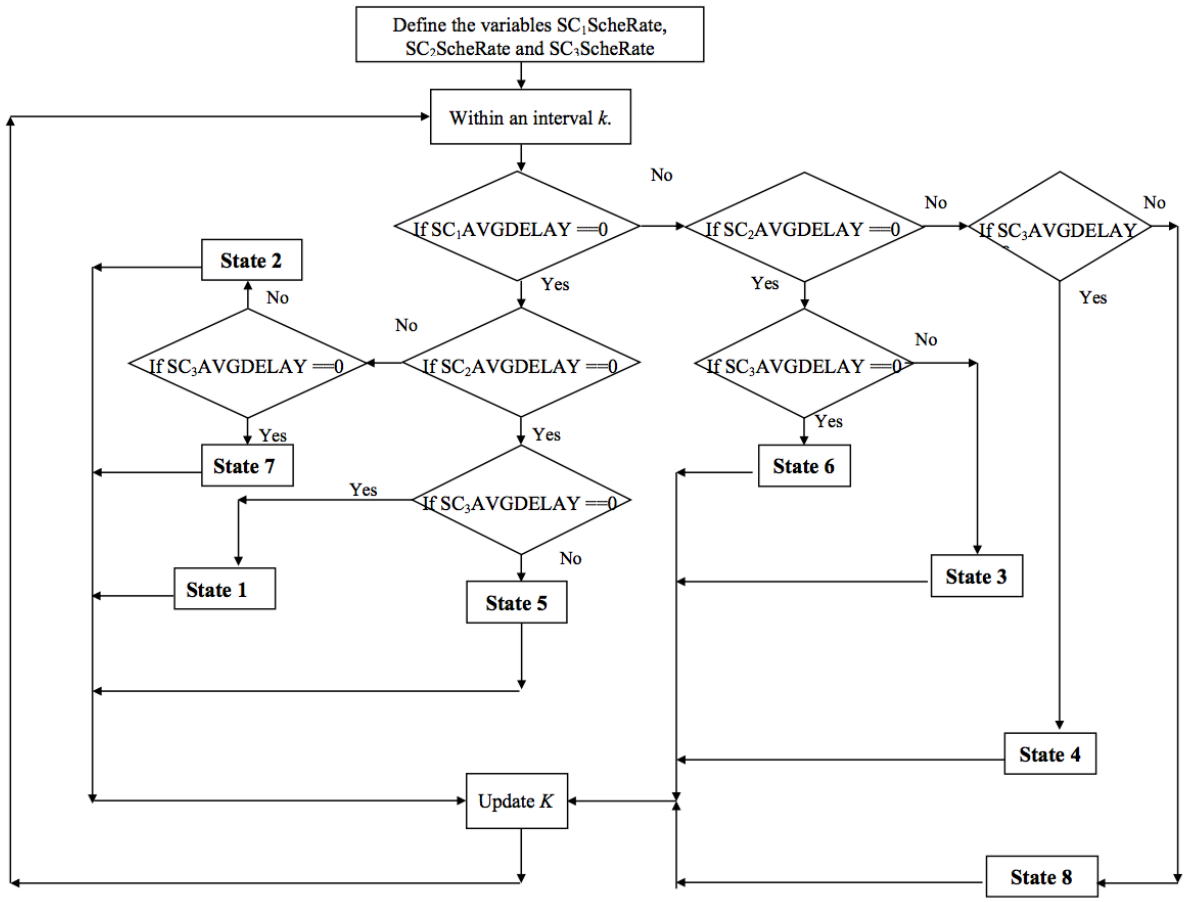
By calculating the average service class queue lengths and delays within an interval *k*, the scheduling rates for service class queues can easily be calculated using Equation 4-8 as described in the previous chapter. If the DiffServ domain supports three service classes, then there are eight possible states in relation to calculating service class scheduling rates. These states are generated based on the availability of packets in the service class queues and are illustrated in Table 5-1. All these states can occur, but it is not absolutely necessitated that all these states do occur during a simulation. For each of the states mentioned in Table 5-1, there is a set of delay differentiation parameters (δ) which are defined between service classes ($\delta_1 : \delta_2 : \delta_3$).

Table 5-1, States for calculating service classes scheduling rates.

State	Description	SC1 Scheduling Rate	SC2 Scheduling Rate	SC3 Scheduling Rate
1	No packets in all queues	✗	✗	✗
2	No packets in SC1 queue	✗	✓	✓
3	No packets in SC2 queue.	✓	✗	✓
4	No packets in SC3 queue.	✓	✓	✗
5	No packets in SC1 and SC2 queues	✗	✗	✓
6	No packets in SC2 and SC3 queues	✓	✗	✗
7	No packets in SC1 and SC3 queues	✗	✓	✗
8	All queues have packets	✓	✓	✓

Within an interval k, the scheduling rates for the service classes in each state are calculated based on the defined ($\delta_1 : \delta_2 : \delta_3$) set, the average service class queue lengths and the delays associated with that state. If there are no packets in one particular service class queue, then the scheduling rate for that queue is set to 0 and the other two classes share all the out-link resources of the DiffServ router, according to their configurable priorities ($\delta_1 : \delta_2 : \delta_3$). If any two service class queues have zero packets, then the out-link resources of the DiffServ router are allocated in entirety to the queue that does have packets. Figure 5-7 illustrates the flow chart for calculating the scheduling rates for service class queues where the DiffServ domain supports three service classes and the code of this process is shown in page 171 of Appendix A-3.

If the DiffServ domain supports three service classes (SC_1,SC_2,SC_3)



State	Description	State	Description
1	Set δ_1: δ_2: δ_3 to 0:0:0 SC1ScheRate = SC2ScheRate = SC3ScheRate =0	5	Set δ_1: δ_2: δ_3 to 0:0:1 SC1ScheRate = SC2ScheRate = 0 SC3ScheRate =SC3CurrUsage/SC3AVGDELAY
2	Set δ_1: δ_2: δ_3 to 0:1:2 SC1ScheRate = 0, SC2ScheRate = SC2CurrUsage/SC2AVGDELAY, SC3ScheRate =0.5* SC2ScheRate *SC3CurrUsage/ SC3CurrUsage	6	Set δ_1: δ_2: δ_3 to 1:0:0 SC2ScheRate = SC3ScheRate = 0 SC1ScheRate =SC1CurrUsage/SC1AVGDELAY
3	Set δ_1: δ_2: δ_3 to 1:0:2 SC1ScheRate = SC1CurrUsage/SC1AVGDELAY, SC2ScheRate =0, SC3ScheRate =0.5* SC1ScheRate *SC3CurrUsage/ SC1CurrUsage	7	Set δ_1: δ_2: δ_3 to 0:1:0 SC1ScheRate = SC3ScheRate = 0 SC2ScheRate =SC2CurrUsage/SC2AVGDELAY
4	Set δ_1: δ_2: δ_3 to 1:2:0 SC1ScheRate = SC1CurrUsage/SC1AVGDELAY, SC2ScheRate = 0.5* SC1ScheRate *SC2CurrUsage/ SC1CurrUsage, SC3ScheRate =0.	8	Set δ_1: δ_2: δ_3 to 1:2:4 SC1ScheRate = SC1CurrUsage/SC1AVGDELAY, SC2ScheRate = 0.5* SC1ScheRate *SC2CurrUsage/ SC1CurrUsage, SC3ScheRate =0.25* SC1ScheRate *SC3CurrUsage/ SC1CurrUsage.

Figure 5-7, flow chart illustrating the process of calculating the scheduling rates for a DiffServ domain which has three service classes.

5.3.6 Calculating Service Class Weights:

After calculating the scheduling rates of the service class queues of the edge routers of a DiffServ domain, the weights of the service class queues of the edge and core routers of a DiffServ domain should also be determined for an interval *k*.

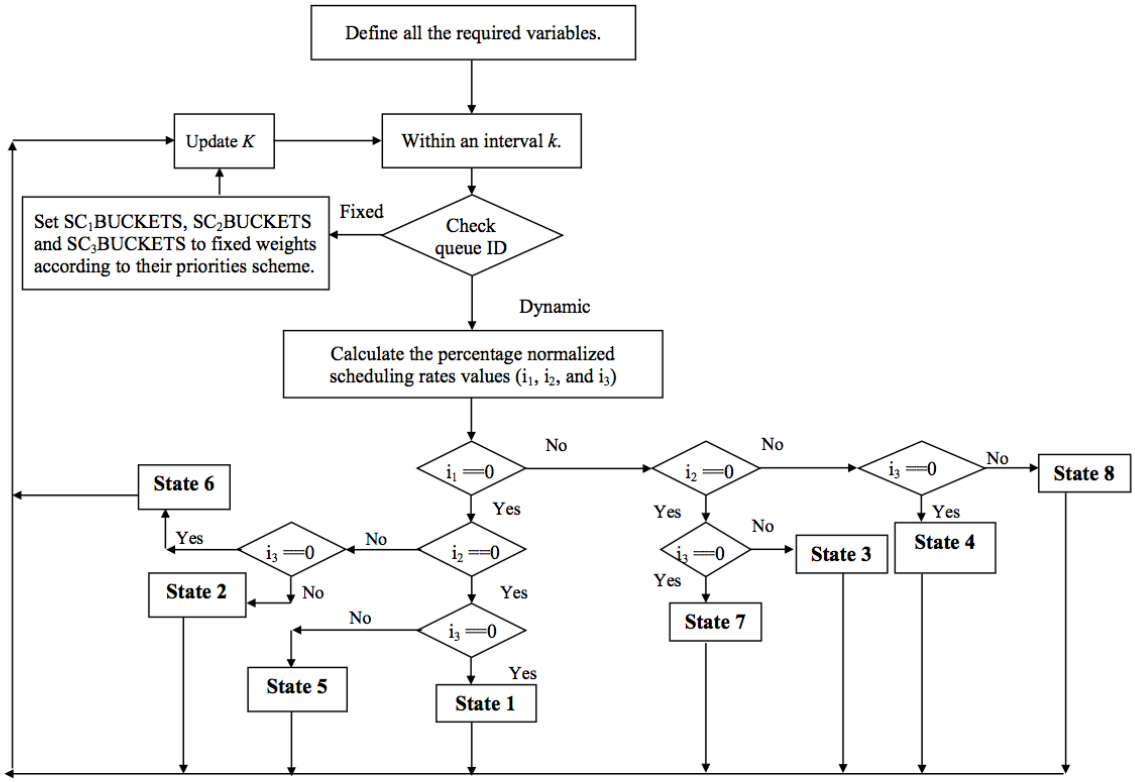
The function (RDWQueue::CalculateBucket) is used to determine the weights of service classes. The weight of a service class queue represents the share of the out–link resources of the DiffServ router that the service class queue can command. The queue weight is expressed, in this research, as the number of packets that are to be forwarded from a queue in a time interval k. The edge routers have dynamic weights while the core router have fixed weights which are pre-configured by the domain administrator. For a DiffServ domain that support three service classes, according to the priorities of these service classes, the fixed weights of core routers could be configured. The queue ID is used to distinguish between the methods for setting the weights of service class queues. Dynamic weights can be calculated using the mathematical Equation 4-13 and Equation 4-14 that have been described in the previous chapter. The normalized percentage scheduling rate values and their Greatest Common Divisor (GCD) and the average packet size of service class queues must also be calculated for an interval *k*. Figure 5-8 shows a flow chart for determining the service classes weights of a DiffServ domain that supports three service classes and the code of this process is shown in page 173 of Appendix A-3.

As in the procedure for calculating service classes' scheduling rates, there are eight possible states in which service class weights must be determined. These states are based on the normalised percentage scheduling rate values. If one service class queue has a zero normalised value, then its weight is set to zero and the other two classes share the out- link resources of the DiffServ router according to their normalised values. If any two service class queues have zero normalised values, then their weights are set to zero while the third service class occupies the whole out-link resource. All these eight states can occur but it is not absolutely necessitated that all these states do occur during the simulation. The states are illustrated in Table 5-2

Table 5-2, States related to calculating service classes weights.

State	Description	SC1 Weight	SC2 Weight	SC3 Weight
1	All queues have zero normalized values	×	×	×
2	SC1 queue has zero normalized value	×	✓	✓
3	SC2 queue has zero normalized value	✓	×	✓
4	SC3 queue has zero normalized value	✓	✓	×
5	SC1 and SC2 queues have zero normalized values	×	×	✓
6	SC1 and SC3 queues have zero normalized values	×	✓	×
7	SC2 and SC3 queues have zero normalized values	✓	×	×
8	All queues have normalized values	✓	✓	✓

If the DiffServ domain supports three service classes (SC_1,SC_2,SC_3)



State	Action	State	Action
1	Set GCD value to 1. Set SC1BUCKETS =0, SC2BUCKETS = 0 and SC3BUCKETS =0.	5	Set GCD value to i3. Set SC1BUCKETS =SC2BUCKETS = 0. Calculate average packets size in SC_3 queue (sc3pktlth). SC3BUCKETS= Out link capacity/ (100*8* sc3pktlth).
2	Calculate GCD (i2, i3). Calculate average packets size in SC_2 and SC_3 queues (sc2pktlth and sc3pktlth) respectively. Set SC1BUCKETS =0. SC2BUCKETS= (i2/GCD) * [Out link capacity/ (100*8* sc2pktlth)]. SC3BUCKETS= (i3/GCD) * [Out link capacity/ (100*8* sc3pktlth)].	6	Set GCD value to i2. Set SC1BUCKETS =SC3BUCKETS = 0. Calculate average packets size in SC_2 queue (sc2pktlth). SC2BUCKETS= Out link capacity/ (100*8* sc2pktlth).
3	Calculate GCD (i1, i3). Calculate average packets size in SC_1 and SC_3 queues (sc1pktlth and sc3pktlth) respectively. Set SC2BUCKETS =0. SC1BUCKETS= (i1/GCD) * [Out link capacity/ (100*8* sc1pktlth)]. SC3BUCKETS= (i3/GCD) * [Out link capacity/ (100*8* sc3pktlth)].	7	Set GCD value to i1. Set SC2BUCKETS =SC3BUCKETS = 0. Calculate average packets size in SC_1 queue (sc1pktlth). SC1BUCKETS= Out link capacity/ (100*8* sc1pktlth).
4	Calculate GCD (i1, i2). Calculate average packets size in SC_1 and SC_2 queues (sc1pktlth and sc2pktlth) respectively. Set SC2BUCKETS =0. SC1BUCKETS= (i1/GCD) * [Out link capacity/ (100*8* sc1pktlth)]. SC2BUCKETS= (i2/GCD) * [Out link capacity/ (100*8* sc2pktlth)].	8	Calculate GCD (i1, i2, i3). Calculate average packets size in SC_1, SC_2 and SC_3 queues (sc1pktlth, sc2pktlth and sc3pktlth) respectively. SC1BUCKETS= (i1/GCD) * [Out link capacity/ (100*8* sc1pktlth)]. SC2BUCKETS= (i2/GCD) * [Out link capacity/ (100*8* sc2pktlth)]. SC3BUCKETS= (i3/GCD) * [Out link capacity/ (100*8* sc3pktlth)].

Figure 5-8, flow chart illustrating the procedure for calculating the weights of service class queues

5.3.7 Configuring Service Classes Scheduling Rates for the Edge Routers of a DiffServ Domain with Multiple Out-Ports:

The service class queues in a DiffServ domain should all have the same management policy at all edge and core routers of the domain in terms of the size of service class buffers, the service class queue dropping probabilities, the service class queue thresholds values, the service class queue scheduling rates for edge routers with multiple out-port links and the fixed forwarding weight for the core routers. This management policy is applied by the domain administrator or by the domain management application. In a DiffServ domain with multi-core routers, the ingress domain routers can have multiple out-port links. At each port, there is a queue for each defined service class: i.e., if a domain has an ingress router with three out-port links, then there must be three queues for a service class (x), one for each of the out-ports of the router, and the scheduling rate for this service class (x) should be the same at all these ports.

The function (RDWQueue::SetSchedulingRateByAverage) is dedicated to the task of setting the same scheduling rate for each service class queue in the ingress routers with multi-out-port links.

In section 4.4, the method for calculating the scheduling rate for a service class queue at an ingress router with multi-out-port links was explained. The function calculates the service class scheduling rate based on the maximum and minimum values. The procedure for calculating the scheduling rates is the same as is explained in section 5.3.5 but with the difference that, here, the maximum and minimum values are used. After the scheduling rate for the specified service class sub queue at each out-port link of the ingress router has been set, the service class weight must then be calculated. The procedure for calculating the service classes' weights is the same as that explained in section 5.3.6. Figure 5-9 illustrates the flow chart of the function for configuring the service classes scheduling rates for the edge routers with multiple out-ports and the code of this function is shown in page 192 of Appendix A-3.

If the DiffServ domain supports three service classes, then x could be SC_1 or SC_2 or SC_3.

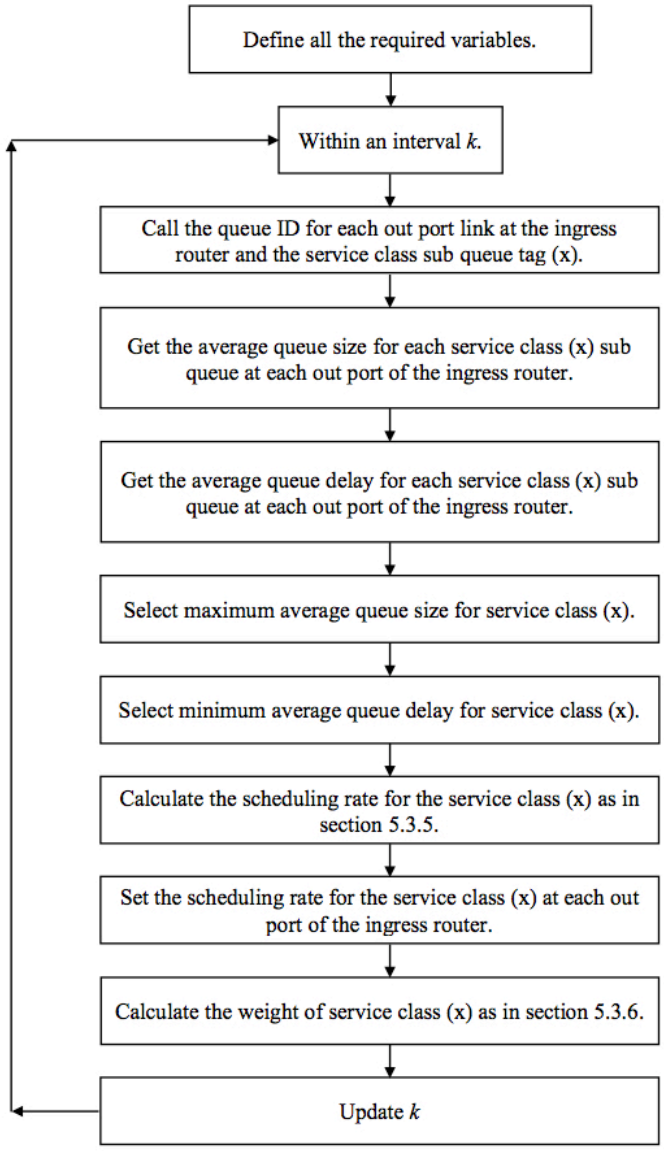


Figure 5-9, flow chart of configuring service classes scheduling rates for the edge routers with multiple out ports function.

5.3.8 Management of Service Classes Scheduling Rates across Different DiffServ Domains:

A function called (RDWQueue::SetSchedulingRateByLinearFactor) has been created in order to manage the service class resources (scheduling rates) across different proportional delay DiffServ in the event of congestion within an interval (*k+1*). The average service class queue length (in bytes) at the ingress router of the downstream domain can be obtained by the use of the function (GetAverageQueueLength1, which is included in the RDWQueue application and used regardless of whether the ingress router has one or many out-ports). This function calculates the congestion level (α) for each service class queue and specifies the scheduling rate update factor (β) for the equivalent service class queue at the egress router of the upstream

DiffServ domain. All these operations are based on the mathematical expressions which are relevant to them as explained in section 4.4 of the previous chapter. After these calculations, the updated scheduling rates for the service class queues should be set and their weights should also be calculated as has already been described in section 5.3.6. Figure 5-10 illustrates the flow chart for the management of service classes scheduling rates across different DiffServ domains and the code of this function is shown in page 196 of Appendix A-3.

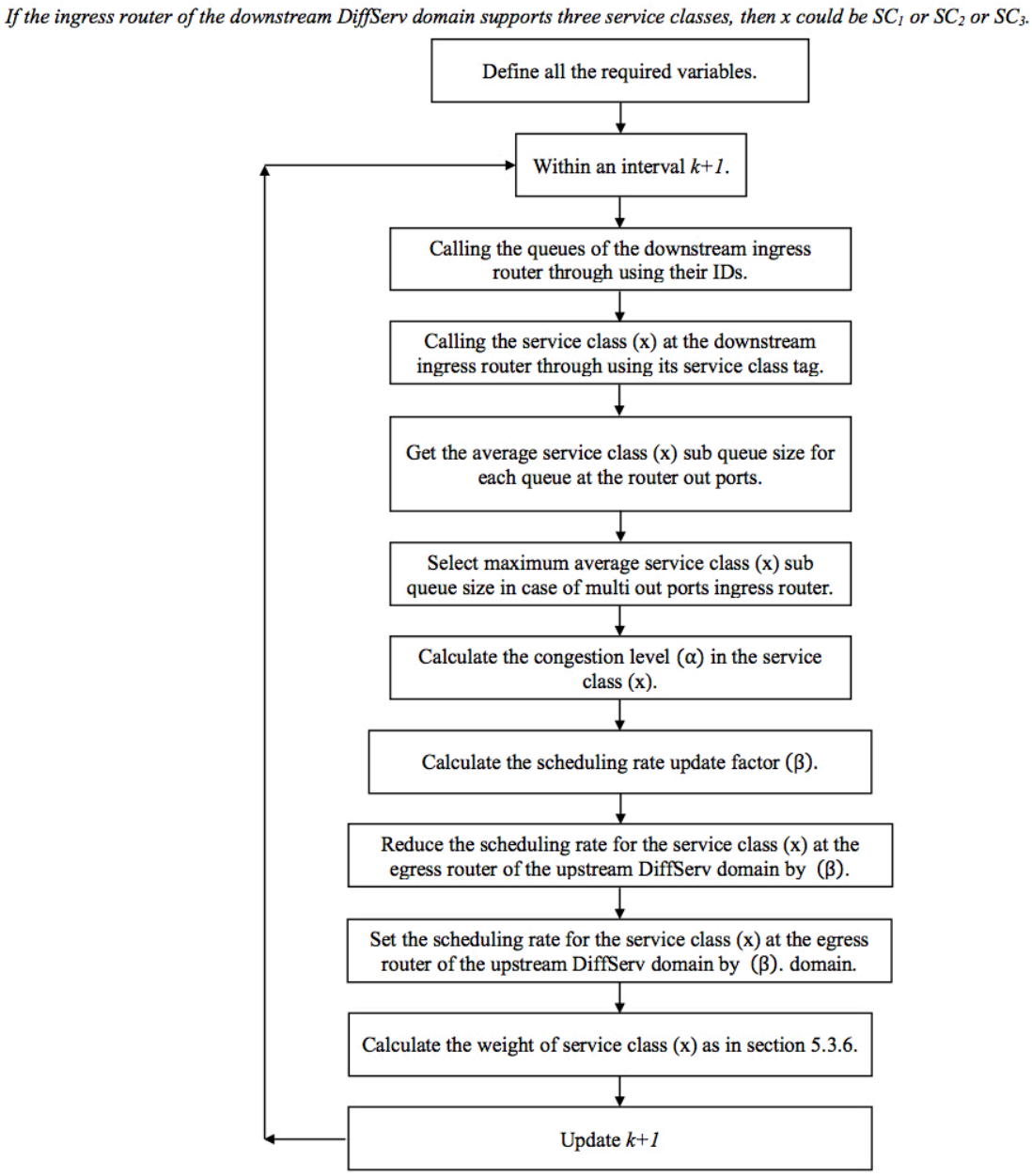


Figure 5-10, flow chart illustrating the function for the management of service classes' scheduling rates across different DiffServ domains.

5.4 OpenFlow Queue Message Layer:

The Queues of OpenFlow switches cannot be configured through OpenFlow flow protocol; however, OpenFlow Management and Configuration Protocol, OF-CONFIG (Open Networking Foundation (ONF) 2014a), which is being standardized by ONF, is used by a controller to configure the queues of OpenFlow switches and it requires OpenFlow 1.2 or later versions. The OF-CONFIG protocol configures the minimum and maximum rates of queue. Each OpenFlow message begins with the same header structure. Figure 5-11 illustrates the message header structure in OpenFlow (Open Networking Foundation (ONF) 2013) (Flowgrammable Research Team 2013).

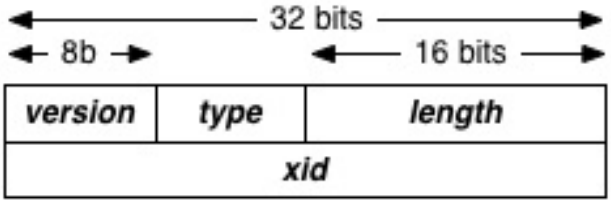


Figure 5-11, OpenFlow message structure.

The version field indicates the version of OpenFlow to which this message belongs. The length field indicates where this message will end in the byte stream starting from the first byte of the header. The xid, or transaction identifier, is a unique value used to match requests to responses while the type field indicates what type of message is present and how to interpret the payload. The controller uses the OpenFlow protocol to query the state of queues through exchanging two messages "QueueGetConfigReq" and "QueueGetConfigRes" to request a queue state and to respond to a controller respectively. Figure 5-12 shows the structure of these mentioned messages. The request message consists of the standard header followed by a 2 byte port identifier, and a 2 byte pad. The response message consists of the standard header, followed by the two byte port identifier, a six byte pad, and a sequence of queues properties.

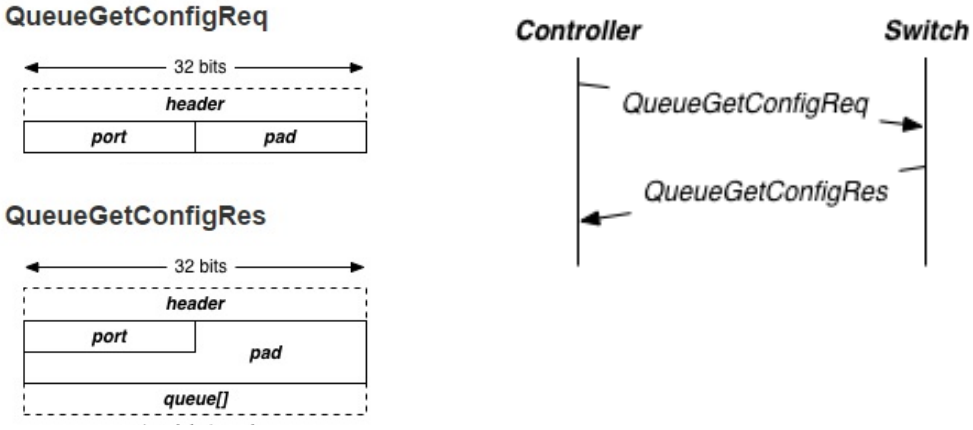


Figure 5-12, OpenFlow queue state query messages.

5.5 Chapter Summary:

This chapter presented the NS3 prototype implementation of the DRAM-NFV scheduling algorithm for evaluation. The reasons for the selection of this specific type of simulation are that NS3 supports the implementation of IP networks and it is open source software that does not incur any financial costs. The methodology implemented is based on the mathematical model of the DRAM-NFV algorithm which was explained in section 4.4. All the NS3 applications mentioned in this chapter and the next chapter (RDWQueue, TosApp, and topologies) are constructed from scratch using the API features of NS3, the C++ programming language and the Linux shell code to analyse the results of the output file. Future work may be pursued by developing some test scenarios to validate and evaluate the DRAM-NFV algorithm implemented using the NS3 simulator; this will be discussed in the next chapter.

Chapter Six
The DRAM-NFV Algorithm:
Validation and Critical Evaluation

6.1 Introduction:

This chapter explains the validation, evaluation, and the results generated by, the DiffServ module and the prototype DRAM-NFV algorithm implemented under the simulation software, NS3. The main goal of this evaluation is to determine whether the DRAM-NFV algorithm can provide better performance (than currently-used algorithms) in managing resources within and across proportional delay DiffServ domains in the event of congestion. Test scenarios have been presented in this chapter which facilitate the study and evaluation of the performance of the DRAM-NFV algorithm. These scenarios take into consideration different topologies of DiffServ domains and study the performance of the DRAM-NFV algorithm in the circumstance that some of the DiffServ domains cannot manage their resources via the NFV technology. The performance of the DRAM-NFV algorithm may be determined in terms of link utilization and average End to End Delay– in relation to these test scenarios. Furthermore, the results in these scenarios are compared with the results to be obtained by the use of the DWFQ algorithm, which manages resources only within the edge routers of the DiffServ domains. The criterion for evaluating the DRAM-NFV is that it should increase the traffic of certain service class queues within the implemented DiffServ module. The DRAM-NFV algorithm monitors the service class congestion levels at the ingress router of the downstream domain and manages resources across the implemented DiffServ domains. The evaluation will proceed by monitoring which service class queues suffer from congestion and measuring the average End to End Delay for service classes traffic and the utilization at the destination link and at the link that connects DiffServ domains. The DRAM-NFV algorithm is applied under the same conditions as the DWFQ algorithm, and the average End to End Delays and link utilizations are measured for both algorithms.

6.2 Traffic Model Simulation Test Applications:

The application module in the network simulator NS3.23 contains several built-in applications that can be used with representations of network topologies. These applications are as follows.

1. OnOff Application – this application generates traffic (TCP/UDP) according to an alternating On/Off pattern. Packets are sent at a constant rate but only in an on period.
2. PacketSink Application – this application consumes traffic (TCP/UDP) sent to an IP address and port. It is considered to be a complement to the OnOff Application but it can be used as a general- purpose application.
3. UdpEchoClient Application – this application transmits a UDP packets to a remote server which is echoed by the remote server and then received (back) by this application.
4. UdpEchoServer Application – this application receives a UDP packet from a remote client and sends it back to that remote client. It is considered to be a complement to the UdpEchoClient application.
5. UdpClient Application – this application transmits UDP packets carrying a 32-bit sequence number and one 64-bit time stamp.
6. UdpServer Application – this application receives UDP packets from a remote host and uses the sequence number to determine if any packets have been lost and the timestamp to determine the packet delay. It is considered to be a complement to the UdpServer application.

The application module of NS3.23 does not provide built-in applications which represent the profiles of real-time applications traffic such as that produced by the Voice over Internet Protocol (VoIP), Video traffic, the File Transfer Protocol (FTP), or Database request messages, etc.

A new application traffic model was created for this research to represent the profile of these abovementioned application traffic flows. This new application is called (TosApp). The NS3::Application class reference is considered as the base class for NS3 applications. All other traffic applications must be derived from this base class; thus the (TosApp) is inherited or derived from the NS3::Application class reference in order to provide a consistent way to start and stop this application. The (TosApp) application defines also the traffic characteristics in terms of the size of the generated packets, the number of generated packets, the rate of generated packets, the packet Type of Service (ToS) and the simulation scheduling of the event of sending a packet. Appendix A-2.2 illustrates the NS3 class references which are used to build the TosApp model and their purposes.

The number of generated packets (nPackets) represents the product of multiplying the simulation run time and the number of packets that must be forwarded in one second from a traffic source. This can be calculated using Equation 6-1:

$$\text{nPackets} = \left[\frac{\text{Traffic Rate (bit/sec.)}}{\text{Packet size (Byte)} * 8}\right] * \text{Simulation stop time}$$

6-1

The time at which the packet will be sent from the source is also calculated by this (TosApp) application by the use of the current simulation time, this time will be used to measure the End to End Delay of each service class traffic transmission.

The profiles of four standard application traffic sources can be used when generating packets in the NS3 simulation. These are implemented as the Voice over Internet Protocol application (VoIP), the Video traffic application, the File Transfer Protocol (FTP) application and the Database request messages application. The characteristics of these applications and their specified queues are shown in Table 6-1 and are explained in (Wendell Odom 2005) (Tim Szigeti 2005). These profiles are used in the simulated traffic model (TosApp) application to generate traffic appropriate to them. The code for (TosApp) traffic model is shown in Appendix A-4.

Table 6-1, Characteristics of the simulated traffic applications profiles.

Application	Service Class	Packet Size (Bytes)	Traffic Rate (Kbits/sec)	Packet rate (Packets/sec)	Inter-arrival time
VoIP	SC1	200 (Constant Distribution)	80	50 (constant)	Constant
Video Traffic	SC2	1300 –1500 (Exponential Distribution)	384	34 (Average)	Random
FTP	SC3	200 – 500 (Exponential Distribution)	256	91 (Average)	Random
Database Request	Any class (Best Effort)	512 (Constant Distribution)	82	20 (Average)	Random

6.3 Experimental Design of the evaluation process:

In order to test and evaluate the performance of the DRAM-NFV algorithm in managing resources within and across different proportional delay DiffServ domains, different testing scenarios should be considered. In these test scenarios, the DiffServ domains topology can be constructed from either single core DiffServ domains or multi-core DiffServ domains. The marking strategies of all the DiffServ domains in all the test scenarios have been made to be identical in order to avoid increasing the complexity of the simulation programming. In addition: all the queues which belong to a specific service class in all DiffServ domains of test scenarios are assumed to have the same queue configuration in terms of queue buffer size (Wendell Odom 2005) (Chin-Chang Li et al. 2000), threshold parameters and drop probabilities for the packet drop algorithm (Cisco 2013); and the same threshold congestion parameters in relation to the management of resource between different proportional delay DiffServ domains. The threshold congestion parameters are based on queue threshold values and its buffer size. Three test scenarios are presented in this research as will be discussed in the following subsections; the procedures for simulating the topologies of these test scenarios using NS3 are explained in Appendix A-5.

The principle of the DRAM-NFV algorithm was explained in chapter 4. The resource management which is carried out across different proportional delay DiffServ domains is based on two factors. Firstly, reducing the scheduling rates of the congested service class queues at the egress router of the upstream domain based on the corresponding service class queue congestion levels at the ingress router of the downstream domain. Secondly, reallocating the resources of the link that connects the DiffServ domains, based on the updated scheduling rates. The algorithm is particularly effective when both the DiffServ domains (upstream and downstream) are underutilized but some of the service class queues at the ingress router of the downstream domain are either congested or suffering from various levels of congestion while other classes are not congested. Consequently, several case studies are presented in relation to each test scenario; each case study includes increasing the traffic of (a) particular service class(es) at the upstream domain to create congestion states in the service class queues at the ingress routers of the downstream domain. The performance of the DRAM-NFV algorithm in managing resources across different DiffServ domains is compared with the performance of the DWFQ algorithm which manages resources within the edge routers of a DiffServ domain but cannot manage resources across DiffServ domains. This performance will be measured, as explained in the section 6.4, by looking at the utilization of the link that connects the DiffServ

domains, the utilization of the link that connects the downstream domain to the destination and the average End to End Delay for service classes' traffic.

6.3.1 Test Scenario 1:

In test scenario 1, both the upstream and downstream DiffServ domains are constructed around single core DiffServ routers. Three case studies are used to test the performance of the DRAM-NFV algorithm in managing resources within and between DiffServ domains in the event of congestion. The cases are as follows:

1. Case study 1 was instituted by increasing the traffic of the medium priority service class (SC_2) at the upstream domain and studying the effects of this on the management of the resources for the lowest priority service class traffic (SC_3) and the highest priority service class traffic (SC_1) - across DiffServ domains when congestion occurs. The traffic of service class (SC_2) is UDP traffic and competes with the TCP traffic (the lowest priority service class) to occupy more bandwidth. The analysis of this case study is shown in section 6.6.1.
2. Case study 2 was instituted by increasing the traffic of the medium priority service class (SC_2) and the lowest priority service class (SC_3) at the upstream domain and studying the effects of this on the management of the resources for the highest priority service class (SC_1) traffic, across DiffServ domains when congestion occurs. The analysis of this case study is shown in section 6.6.2.
3. Case study 3 was instituted by increasing the traffic of all the service classes at the upstream domain to study how the DRAM-NFV algorithm achieves better utilizations for the service classes at the link that connects the DiffServ domains – when congestion occurs. The analysis of this case study is shown in section 6.6.3.

The topology for test scenario 1 is shown in Figure 6-1. The topology simulation diagram and the NS3 code for this scenario are presented in Appendix A-5.1.

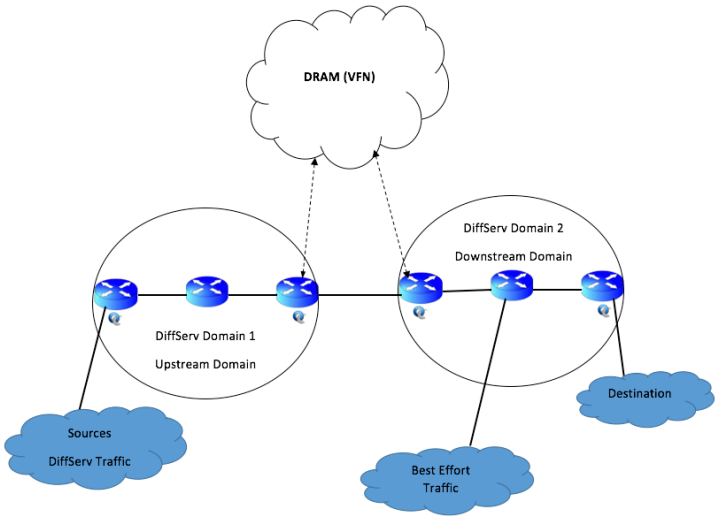


Figure 6-1, Test scenario 1.

6.3.2 Test Scenario 2:

in test scenario 2, the upstream DiffServ domain is constructed around a single core DiffServ router but the downstream DiffServ domain is constructed around multi core DiffServ routers. Two case studies were carried out to test the performance of the DRAM-NFV algorithm in managing resources within and between DiffServ domains when congestion occurs. These cases were:

1. Case study 1 was instituted by increasing the traffic of the lowest priority service class (SC_3) at the upstream domain to study the performance of the DRAM-NFV algorithm in managing the resources for the highest priority service class traffic (SC_1) and the medium priority service class traffic (SC_2) across DiffServ domains when congestion occurs. The traffic of service class (SC_3) is TCP-type traffic and it competes with the UDP traffic (the highest and medium priority service classes) to occupy bandwidth. The analysis of this case study is shown in section 6.7.1.
2. Case study 2 was instituted by increasing the traffic of all service classes at the upstream domain to study how the DRAM-NFV algorithm achieves better utilizations for service classes at the link that connects DiffServ domains and at the links connecting the downstream DiffServ domain with multi core routers with destination when congestion occurs. The analysis of this case study is shown in section 6.7.2.

The topology of test scenario 2 is shown in Figure 6-2. The topology simulation diagram and the NS3 code for this scenario are given in Appendix A-5.2.

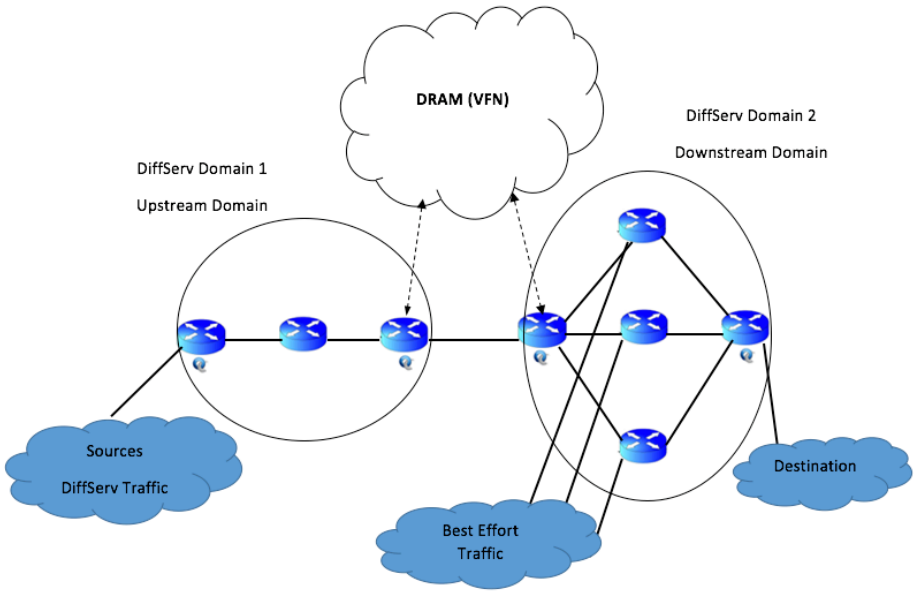


Figure 6-2, Test scenairo2.

6.3.3 Test Scenario 3:

In test scenario 3, there are two upstream DiffServ domains and one downstream DiffServ domain; the upstream domains are constructed around single core DiffServ routers while the downstream DiffServ domain is constructed around multi core DiffServ routers. The performance of the DRAM-NFV algorithm will be studied in relation to the situation where there is more than one upstream DiffServ domain and one of these cannot manage its resources via the NFV technology.

In this test scenario, all three case studies were carried out to test the performance of the DRAM-NFV algorithm in managing resources within and among DiffServ domains when congestion occurs. These cases are:

1. Case study 1 is instituted by increasing the traffic of the medium priority service class (SC_2) at one of the upstream domains and studying the effects of this on the managing of the resources for the lowest priority service class traffic (SC_3) and the highest priority service class traffic (SC_1) among DiffServ domains and at the downstream domain, when congestion occurs. The analysis of this case study is shown in section 6.8.1.
2. Case study 2 is instituted by increasing the traffic of the lowest priority service class (SC_3) at the upstream domains to study the performance of the DRAM-NFV algorithm in managing the resources for the highest priority service class (SC_1) traffic between DiffServ domains when congestion occurs. The analysis of this case study is shown in section 6.8.2.

3. Case study 3 is instituted by increasing the traffic of all the service classes at the upstream domain to study how the DRAM-NFV algorithm achieves better utilizations for service class traffic at the link that connects the DiffServ domains - when congestion occurs. The analysis of this case study is shown in section 6.8.3.

The topology of test scenario 3 is shown in Figure 6-3. The topology simulation diagram and the NS3 code for this scenario are given in Appendix A-5.3.

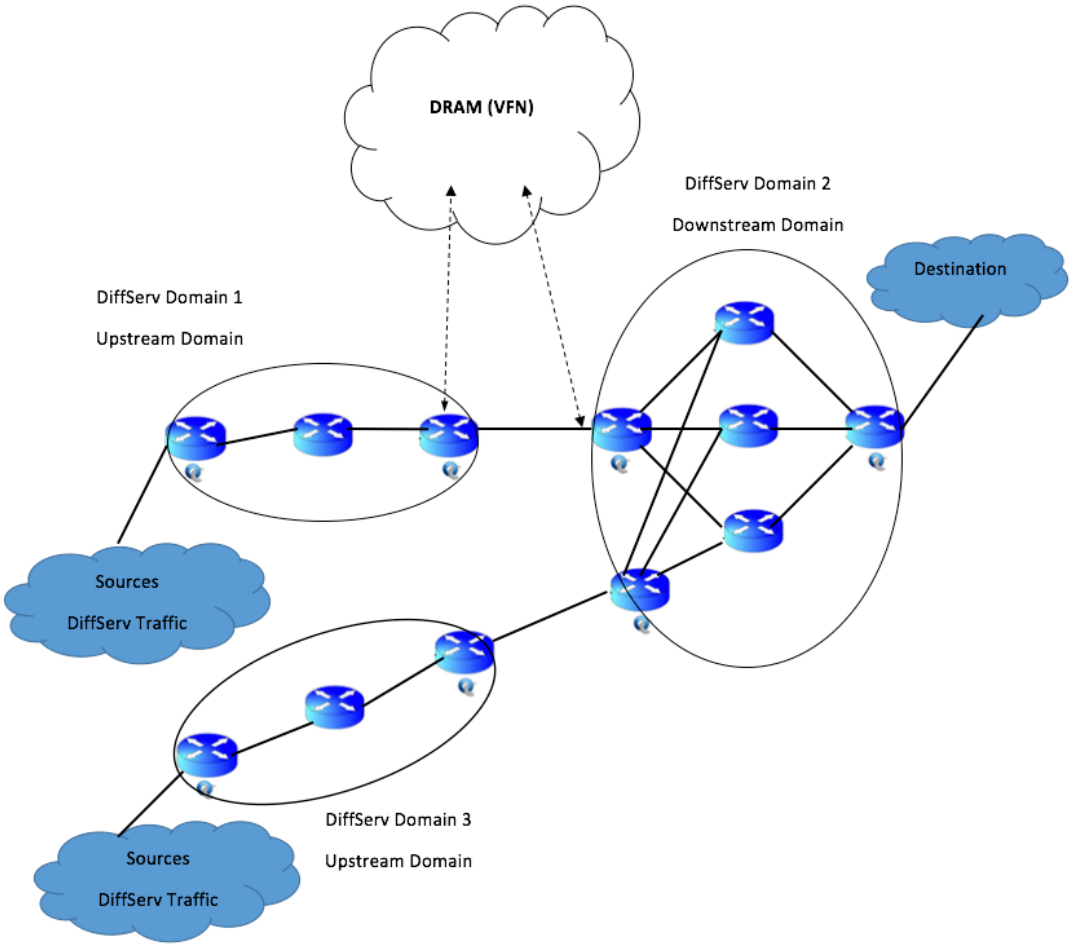


Figure 6-3, Test scenairo3.

6.4 Performance Measurements in Relation to the Simulation of the Differentiated Services Domains:

Two parameters are used to measure the performance of the DiffServ domains. First, there is the average End to End Delay for each service class' traffic and for the overall traffic. The second performance parameter is the link utilization in the DiffServ domains. The End to End Delay of each packet received at the destination node (EtE PktDelay) is calculated based on the difference between the time the packet was sent and the time the packet was received. The average end-to-end delay for a specific service class' traffic ($EtE\ SC_iDelay$) represents the average of the accumulative End to End Delays of packets that are encountered by this service class. The average End to End Delay for the overall traffic ($EtE\ Delay$) represents the average end-to-end delays of all packets received at the destination node. The code for calculating the average End to End Delay for service classes traffic is shown in Appendix A-7.

The percentage utilization related to a service class' traffic at a certain link in the DiffServ domain is the ratio of current service class traffic (bits) at the link port per second to the maximum traffic that the link port can handle. The service class traffic represents the sum of the service class packets sizes in (bits) that are sent from the service class queue across the out-port link in one second. The service class' traffic link utilization measurements are accomplished in the RDWQueue::DoDequeue, RDWQueue::UpdateLinkUtilization and RDWQueue::getLinkUtilization functions. The code for calculating the service classes traffic utilizations is shown in page 160 of Appendix A-3.

All the results of the simulation, in terms of service classes' average queue delay, service classes' average queue length, service classes' scheduling rates, service classes' weights, service classes' link utilizations, packet sending time, packet receiving time, the average End to End Delay values, the measured congestion level values, the scheduling rate update factor values and other indicators are all stored in a (.txt) file during the running of the simulation. This (.txt) file needs to be put through a filtering process in order to extract the abovementioned values separately. The code for setting the values of simulation parameters and the code script for obtaining the simulation results from the execution file (.txt) are shown in Appendixes A-6 and A-8 respectively.

6.5 Analysis of the Simulated Proportional Delay DiffServ Queue model:

In this section, the performance of the simulated proportional delay DiffServ queue model will be discussed in relation to the situation where the DRAM-NFV algorithm is not in use. For this analysis the DWFQ algorithm is used, and the traffic condition in the service class queues is underutilisation. The discussion covers the following issues:

i. configuration of the service class queues;
ii. average service class queue delays;
iii. service class packets drop mechanism.

The proportional delay DiffServ domain is represented by the simulated queue model application (RDWQueue), the traffic model application (TosApp) and the network topology model. The (TosApp) application generates four different standard traffic flow types; these are VoIP, Video traffic , FTP and database request messages. The simulated DiffServ domains are designed to provide three service class queues (SC_1, SC_2 and SC_3), differentiated by the way that their scheduling rates are specified, as illustrated in section 4.3. (Chin-Chang Li et al. 2000) set the delay configuration parameters (δ) for service classes (SC_1:SC_2:SC_3), which are configured by the domain administrator, to (1:2:4) such that SC_1 is the highest service priority and SC_3 is the lowest priority service class. In addition, (Chin-Chang Li et al. 2000) tested the performance of these parameters in achieving the PDD conditions which were explained in 2.2. Based on (Chin-Chang Li et al. 2000) assumption, the delay differentiation parameters for service classes in the simulated DiffServ domain are set as (SC_1:SC_2:SC_3) (1:2:4) respectively such that the scheduling rate for the medium priority (SC_2) queue is half that of the scheduling rate for the highest priority (SC_1) queue and the scheduling rate of the lowest priority (SC_3) queue is half that of the scheduling rate for the medium priority (SC2) queue as illustrated in Equation 4-7 and Equation 4-8.

In relation to the QoS requirements of the abovementioned traffic applications, VoIP traffic quality is significantly affected by packet loss and delay. It should be marked as DSCP EF (J. Babiarz, K. Chan & F. Baker 2006) which has the highest priority as compared to other the classes, according to the standard traffic differentiation. Video traffic should be marked as DSCP AF_{41}, which is the second highest priority class after the EF class, based on the standard traffic differentiation. This traffic is also sensitive to packet delay and loss. FTP traffic is considered to be bulk data traffic. It is relatively insensitive to delay and packet loss, and should be marked as DSCP $AF_{11,}$ based on the standard traffic differentiation. Consequently, the service class (SC_1) in the simulated DiffServ module is allocated to VoIP traffic, the service

class (SC_2) is allocated to Video traffic and the service class (SC_3) is allocated to FTP traffic. The data base request message traffic (SC_4) traffic is considered a best effort traffic. It should be marked as $DSCP_{0,}$ based on the standard traffic differentiation. This kind of traffic can be used to change the network traffic conditions in the downstream DiffServ domain. The best effort traffic should occupy the remaining bandwidth so it is inserted in any defined service class queues (SC_1, SC_2 and SC_3), depending on the available resources in these classes.

The buffer sizes of these abovementioned service class queues can be expressed in terms of power of 2 numbers of packets: i.e., (64, 128, 256, 512) packets (Wendell Odom 2005). These buffer sizes can also be expressed in terms of bytes by multiplying the number of packets that are required to be accommodated in a queue by the average packet size (in bytes) in that queue. The buffer size of each of the abovementioned service class queues (SC_1, SC_2, SC_3) in the simulated DiffServ module (upstream and downstream domains) is configured to accommodate 128 packets respectively. The (WRED) algorithm is used as a queue management technique intended to avoid congestion in the queues. It prevents a queue from ever filling to its buffer size. The threshold parameters and drop probabilities for (WRED) are also configurable parameters and can be set by the administrator of a DiffServ domain. The threshold parameters and drop probabilities for each defined service class queue in the simulated DiffServ module (upstream and downstream domains) are the same. The WRED configuration of the Cisco™ [5] 10000 series QoS routers (Cisco 2013) is used in this simulation, as illustrated in Table 6-2. The threshold values for the congestion level of all the service class queues at the ingress router of the downstream domain are the same. These values are also configured by the domain administrator and are chosen such that the lower congestion threshold value is set to the minimum queue threshold value as indicated in Table 6-2 and the higher threshold congestion value is set to 0.9 to get a wide and linear range for the scheduling rate update factor (β). The relation between the congestion level (α) and the scheduling rate update factor (β) was described in section 4.3.

[5] Cisco™ is a trade mark of Cisco Systems, Inc. and its affiliates.

Table 6-2, configuration of the simulated WRED

	Minimum Threshold (times the queue size)	Maximum Threshold (times the queue size)	Drop Probability
Highest Priority (SC_1)	15/32	½	1/10
Medium Priority (SC_2)	7/16	½	1/10
Lowest Priority (SC_3)	13/32	½	1/10

As the service class (SC_1) is the highest priority service class in relation to the other classes (SC_2) and (SC_3), the performance of this class, in terms of average queue delay, should be better than that of both the SC_2 and the SC_3 queues. It is expected that the SC_1 queue should suffer the least average queuing delay because this queue occupies all the required bandwidth for its traffic. The performance in the service class (SC_2) queue should have an asymptotic behaviour in relation to the service class (SC_1) queue. Its performance should not be better or much worse than that of the SC_1 queue. In contrast, the SC_3 queue should have a markedly different performance to the SC_1 and SC_2 queues; this queue will witness variations in its average queuing delay because it occupies the least bandwidth compared to the other queues. Figure 6-4 and Figure 6-5 illustrate the average queue delays at the ingress and egress routers of the simulated DiffServ domain for the whole period of simulation.

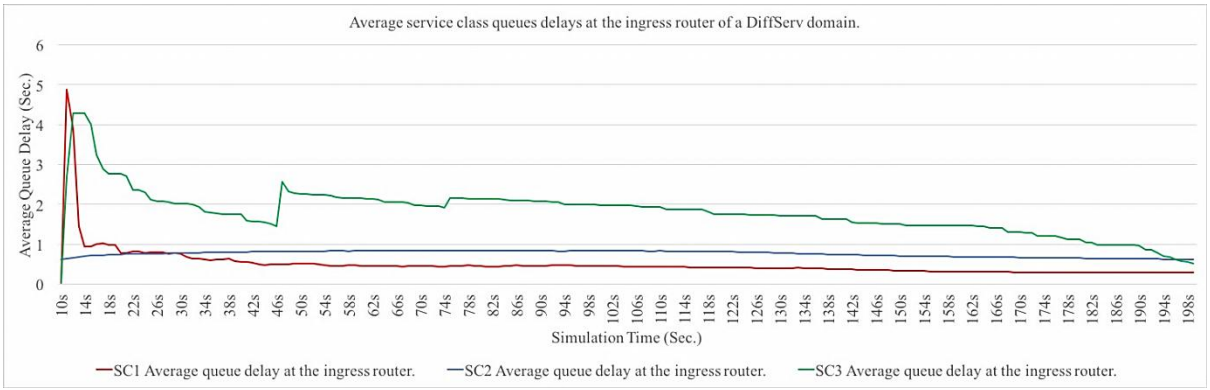


Figure 6-4, Average service class queue delays at the ingress router of the downstream DiffServ domain

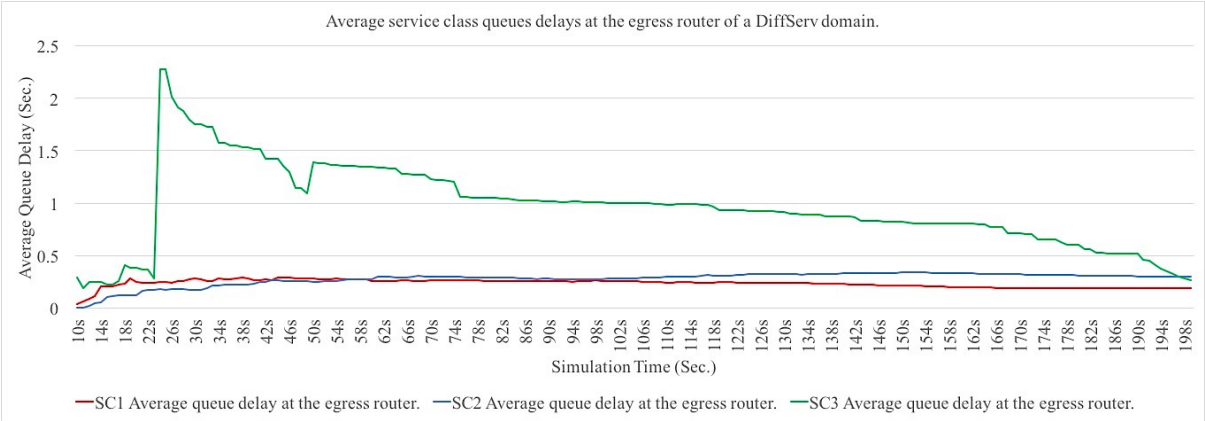


Figure 6-5, Average service class queue delays at the egress router of an upstream DiffServ domain

The ingress and egress routers of the simulated DiffServ domains have their service class scheduling rates adjusted dynamically within an interval *k*. Adjusting the scheduling rates of service classes dynamically maintains the delay differentiation between classes. Table 6-3 illustrates the average service class queue delay ratios and the delay differentiated parameter ratios for the whole period of simulation.

Table 6-3, service classes' average queue delay ratios and their delay differentiated parameter ratios for whole period of simulation – at the ingress router of a downstream DiffServ domain.

	SC1 class	SC2 class	SC3 class
Service class mean delay for the whole period of simulation (sec.)	**0.501**	**1.076**	**2.365**
Service class delay differentiated parameter (δ)	**1**	**2**	**4**

	Service classes mean delays ratio	Service classes delay differentiated parameters ratio (δ)
Class 1: class 2 ratio	**0.465**	**0.5**
Class 2: class 3 ratio	**0.455**	**0.5**
Class 1: class 3 ratio	**0.212**	**0.25**

6.6 Test Scenario 1 Analysis:

In test scenario 1, the upstream and downstream domains both consist of single core routers, as shown in Figure 6-1. The simulation diagram of the test scenario 1 topology is illustrated in appendix A-5.1- Figure 9-8. The configuration parameters of test scenario 1 is shown in Table 6-4. The best effort traffic (SC_4) is injected into the downstream domain through the core routers to change the traffic conditions within that domain. This traffic can be injected into any defined service class queue depending on the available resources in those queues.

Table 6-4, Configuration parameters of test scenario 1.

Parameter	Configuration value	Notes
Link capacities of the upstream domain	2 Mbits	To ensure that more DiffServ traffic is injected into the downstream domain from the upstream domain.
Link capacity that connects the upstream and downstream domains.	4 Mbits	
link capacities of the downstream domain	2 Mbits	
Link capacity that connects the downstream domain and destination.	4 Mbits	
Both the upstream and downstream domains provide three service class queues	(SC_1, SC_2, SC_3)	
The delay configuration parameters (δ) for service classes (SC_1:SC_2:SC_3) at the edge DiffServ routers.	(1:2:4)	(Chin-Chang Li et al. 2000).
The service class markings in the upstream and downstream domains.	Identical.	
Service class queue buffer size in the upstream and downstream domains.	128 packets	(Wendell Odom 2005).
Minimum (lower) Service class queue threshold values (SC_1:SC_2:SC_3) in the upstream and downstream domains.	(15/32:7/16:13/32) times the queue size.	(Cisco 2013)
Service class queue dropping probabilities (SC_1:SC_2:SC_3) in the upstream and downstream domains.	(1/2:1/2:1/2) times the queue size.	(Cisco 2013)
Traffic Distribution in the upstream and downstream domains.	VoIP (SC1) – Highest Priority, Video Traffic (SC2), FTP (SC3) – Lowest Priority, and Data Base Request message (Best Effort).	
Lower threshold congestion values for (SC_1:SC_2:SC_3) in the downstream domain	(15/32:7/16:13/32)	(Cisco 2013)
Higher threshold congestion values for (SC_1:SC_2:SC_3) in the downstream domain	(0.9:0.9:0.9)	To get a linear scale of resources management
Maximum Scheduling rate update factor (β_{max}) for (SC_1:SC_2:SC_3) in the downstream domain.	(0.9:0.9:0.9)	To damp the throughput at upstream domain.

The edge routers in the upstream and downstream domains use dynamic weights to forward their traffic while the core routers use pre-configured fixed weights; they forward packets from their queues based on this sequence (SC_1:SC_2:SC_3) (5:4:3) packets respectively (Dovrolis 2000); it illustrated in an experiment how the weights of WFQ are selected and their effect in achieving controllable delay differentiation between classes.

As explained in section 6.3, to test the performance of the DRAM-NFV algorithm in managing resources within and across different DiffServ domains in test scenario (1), in the event of the traffic of certain service class queues are increased individually and the service class congestion levels at the ingress router of the downstream DiffServ domain is monitored by applying the DWFQ algorithm. In a proportional delay DiffServ domain, all the service class queues share one out link to forward their traffic. The highest priority service class queue is always occupying its required bandwidth for its traffic in the event of congestion while the other classes are competing between them to occupy the remaining bandwidth. This competition is based on the delay differentiated parameters (δ), as explained in chapter 4. Increasing the traffic in any of the non-highest or highest priority service classes can cause congestion in other classes because when the traffic of a service class queue increases, its service classes scheduling rate increases as the scheduling rate is directly proportional to the service class queue length. Consequently, the distribution of all service class weights, which represent the service classes' shares in the out link capacity, can be affected by granting the highest weight for the service class with the highest scheduling rate and the lowest weight for the service class with the least scheduling rate.

In test scenario (1), three case studies are undertaken to test the performance of the DRAM-NFV algorithm. These are as follows:

6.6.1 Case Study 1- Test Scenario 1 Analysis:

In this case study, The traffic of the medium priority service class (SC_2) at the upstream domain of Figure 6-1 is increased - i.e., more SC_2 traffic sources are connected to the DiffServ traffic sources node (n_0) of the simulated test scenario (1) topology, appendix A-5.1- Figure 9-8. The measured congestion levels (α) in the service class queues (Q_{168}) at the ingress router of the downstream DiffServ domain when resources are not managed across the DiffServ domains – i.e., by applying only the DWFQ algorithm – are shown in Figure 6-6, Figure 6-7 and Figure 6-8 (dashed curves) respectively. It is noticeable that the SC_1 and SC_2 queues suffer from different levels of congestion because of the increase in SC_2 traffic, while the SC_3 queue does not. The buffer resources of the SC_1 and SC_3 queues are not well utilized during the

periods of congestion in the SC_2 queue at the ingress router of the downstream domain. This represents the research problem that needs to be solved through the managing of the resources across DiffServ domains.

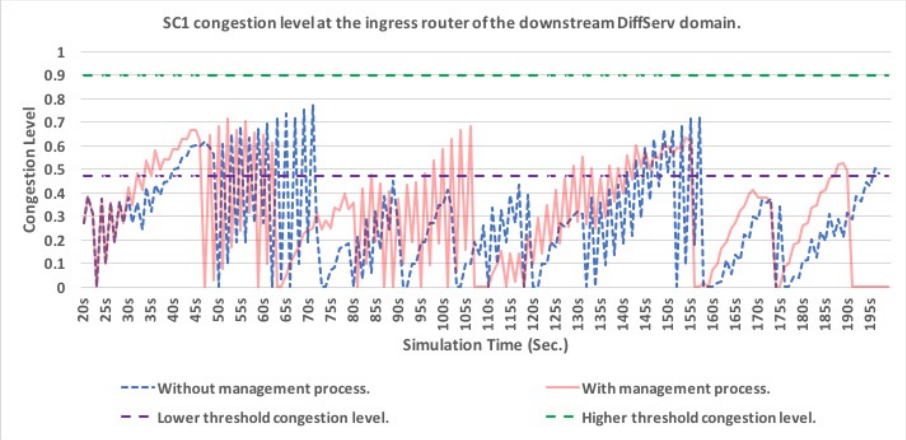


Figure 6-6, SC_1 congestion level - case study 1, test scenario 1

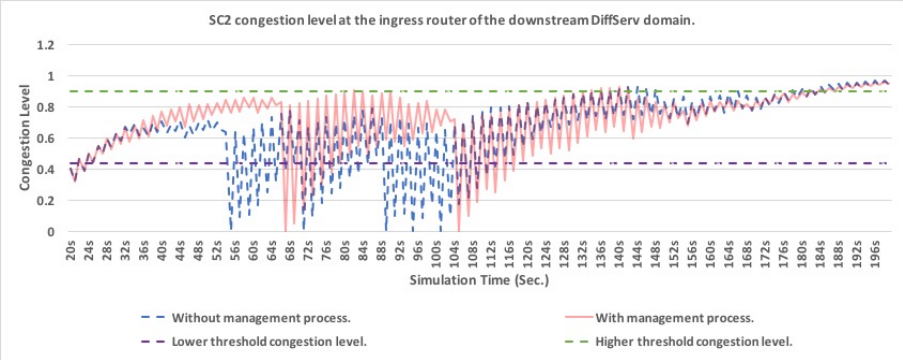


Figure 6-7, SC_2 congestion level - case study 1, test scenario 1

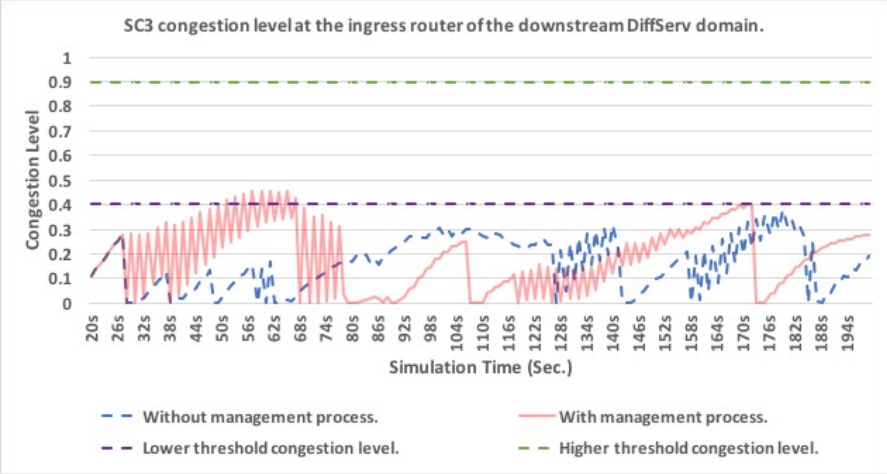


Figure 6-8, SC_3 congestion level - case study 1, test scenario 1

When the DRAM-NFV algorithm is applied to manage resources across the DiffServ domains, the scheduling rates for the service class queues that, before, suffered from congestion at the egress router of the upstream DiffServ domain are reduced. Also the resources of the link that connects the DiffServ domains is redistributed in a different way such that the utilizations of service classes traffic at that link are organised based on the traffic conditions in the ingress router of the downstream domain and the traffic conditions at the egress router of the upstream domain.

In the event of congestion, the DRAM-NFV prevents any class that has a certain level of congestion above the lower threshold congestion level from occupying all the bandwidth it demands and grants some extra resources to the other classes. Because of this process, the link utilization for the highest priority service class (SC_1) is reduced compared to the situation which pertains when only the DWFQ algorithm is applied, as shown in Figure 6-9, while, as shown in Figure 6-10 and Figure 6-11, the utilization of the medium and the lowest priorities service classes (SC_2 and SC_3) respectively are increased. The average percentage utilizations of the service classes at the link that connects the DiffServ domains, in relation to both the DWFQ and the DRAM-NFV algorithms, are shown in Table 6-5 for the whole period of simulation.

Table 6-5, Average percentage utilization for the service classes at the link that connects DiffServ domains - Test scenario 1, case study 1.

Service Class	Domains Link utilization with DWFQ algorithm	Domains Link utilization with DRAM-NFV algorithm
SC1	6.2%	4.72%
SC2	36.98%	38.42%
SC3	0.48%	0.53%

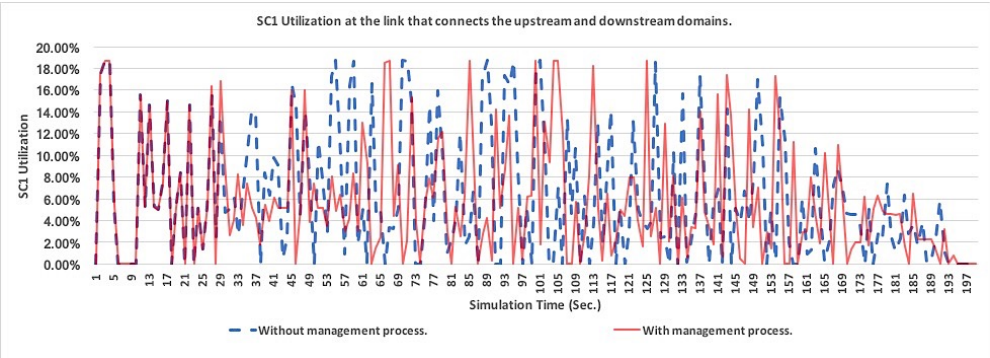


Figure 6-9 , SC_1 utilization at domains link - case study 1, test scenario 1.

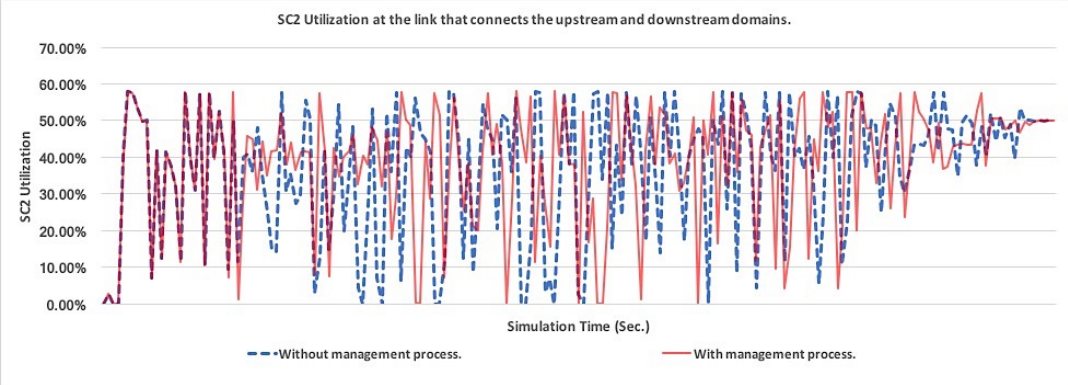


Figure 6-10, SC_2 utilization at domains link - case study 1, test scenario 1.

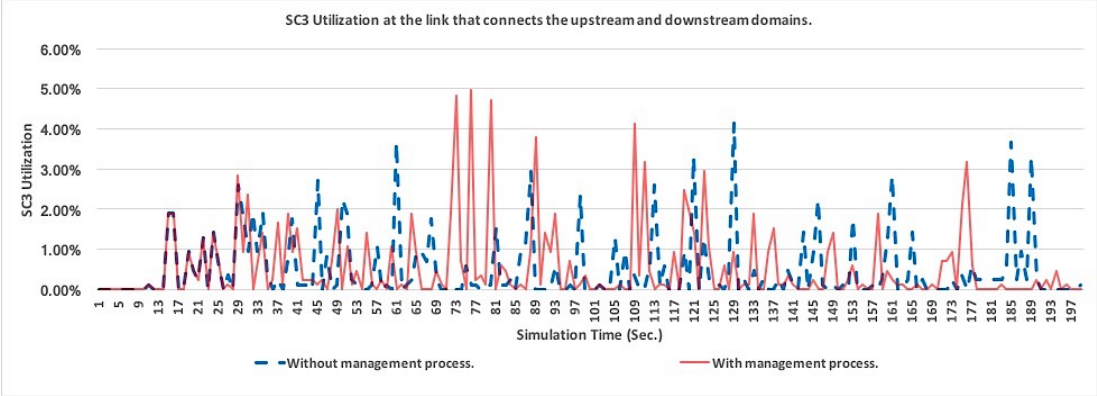


Figure 6-11, SC_3 utilization at domains link - case study 1, test scenario 1.

This improvement in that link's utilization creates also an improvement in the service classes' traffic utilization at the destination as illustrated in

Table 6-6. These differences in service classes' traffic utilization values could be due to the injection of best effort traffic at the downstream domain which disturbs the traffic state at that domain.

Table 6-6, Average percentage utilization for the service classes at the destination link - Test scenario 1, case study 1.

Service Class	Destination Link utilization with DWFQ algorithm	Destination Link utilization with DRAM-NFV algorithm
SC1	1.55%	1.46%
SC2	16.26%	16.49%
SC3	0.428%	0.463%

The average End to End Delay for service classes traffic in the upstream and downstream DiffServ domains is illustrated in Table 6-7 for the whole period of simulation.

Table 6-7, Average End-to-End Delay, test scenario 1 - case study 1.

Service Class	Average End-to-End delay using DWFQ algorithm	Average End-to-End delay using DRAM-NFV algorithm	Increased/ Decreased
SC1	4.01 sec	4.34 sec	Increased by 0.33 sec
SC2	4.29 sec	4.34 sec	Increased by 0.05 sec
SC3	105.77 sec	101.924 sec	Decreased by 3.85 sec
Overall traffic	9.361 sec	8.66 sec	Decreased by 0.7 sec

When comparing the congestion levels of the service class queues at the ingress router of the downstream DiffServ domain Figure 6-6, Figure 6-7 and Figure 6-8. it is noticeable that DRAM-NFV increases the congestion level in these queues at some intervals during the simulation. This means that the DRAM-NFV algorithm participates in improving the service class buffer utilizations at the ingress router of the downstream domain during the periods of congestion in the SC_1 and SC_2 queues.

From Table 6-5, Table 6-6 and Table 6-7; it is noticeable that the DRAM-NFV algorithm achieves better balance for DiffServ domains resources through monitoring the bandwidth hungry service class at the downstream domain and managing its resources at the upstream domain. As a consequence of this, the utilizations of some service class across DiffServ domains are improved and the average End to End Delay for overall traffic is also reduced. Improving the utilizations of service classes traffic and reducing the average End to End Delay for overall traffic means that the DRAM-NFV algorithm participates in improving the QoS of application traffic during the congestion periods.

6.6.2 Case Study 2- Test Scenario 1 Analysis:

In the second case study, the traffic for the medium and lowest priority service classes (SC_2) and (SC_3) of Figure 6-1 respectively are increased; by creating more SC_2 and SC_3 traffic sources at the DiffServ traffic sources node (n_0) of the simulated test scenario (1) topology diagram, appendix A-5.1- Figure 9-8. When applying the DWFQ algorithm only for managing resources within the edge routers of the DiffServ domains, it is noticeable that the SC_2 queue at the ingress router of the downstream domain (Q_{168}) suffers from congestion more than the others, as shown in Figure 6-13 (dashed curve). The SC_1 and SC_3 queues suffer from different levels of congestion, as shown in Figure 6-12 and Figure 6-14 respectively (dashed curves).

When the DRAM-NFV algorithm is applied, it causes a reduction in the congestion levels in the service class queues throughout the conditions pertaining to the periods of congestion that occurred when the DWFQ algorithm was applied, but it increases the congestion level in some intervals that were not congested when the DWFQ algorithm was applied, as shown in Figure 6-12, Figure 6-13 and Figure 6-14. These figures illustrate the congestion levels in the service class queues at the ingress router of the downstream domain when the DWFQ and subsequently the DRAM-NFV algorithms are applied.

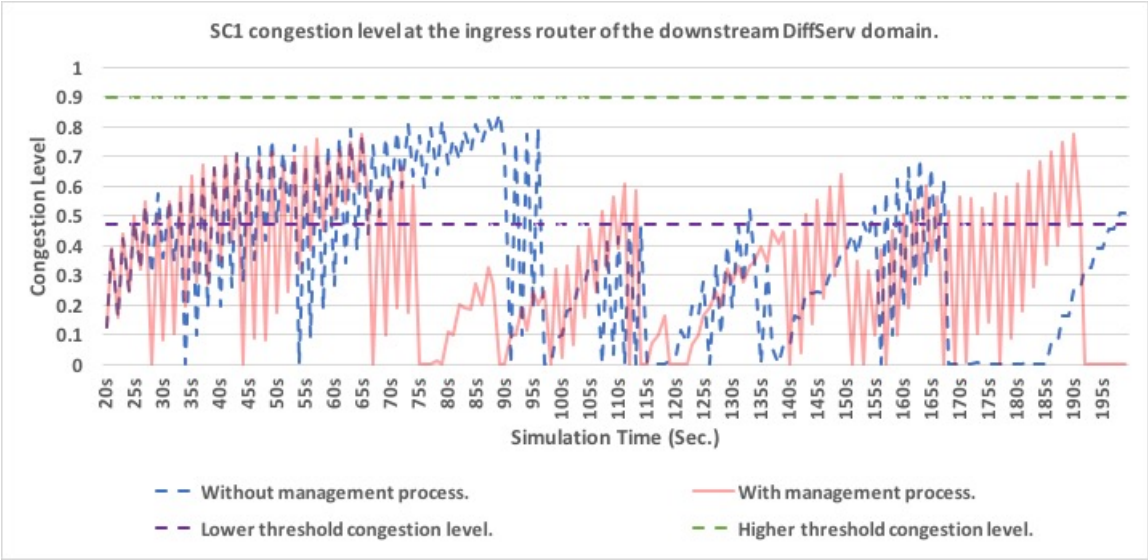


Figure 6-12, SC1 congestion levels in relation to the DWFQ and DRAM-NFV algorithms - case study 2, test scenario 1.

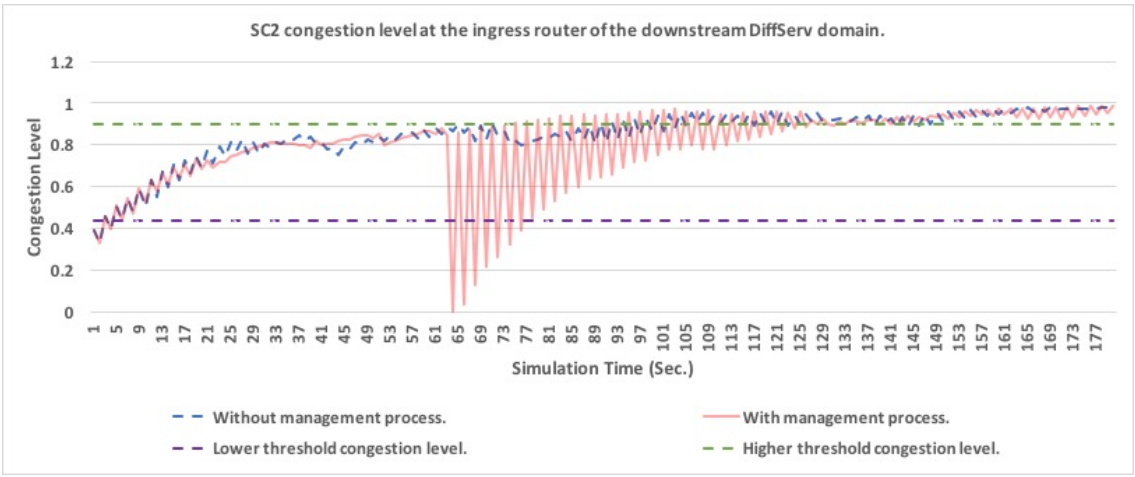


Figure 6-13, SC2 congestion levels in relation to the DWFQ and DRAM-NFV algorithms - case study 2, test scenario 1.

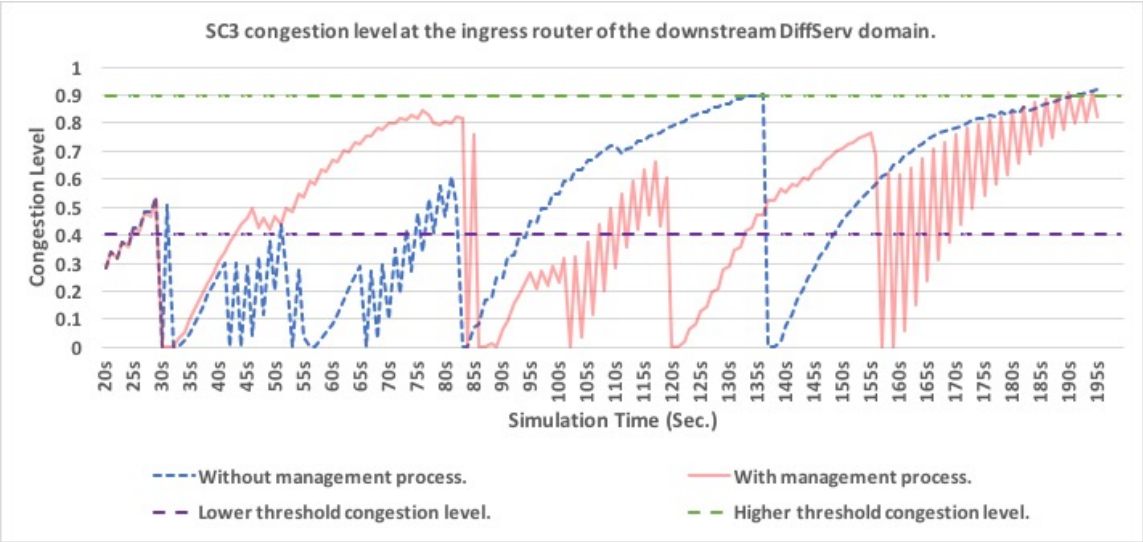


Figure 6-14, SC3 congestion levels in relation to the DWFQ and DRAM-NFV algorithms - case study 2, test scenario 1.

The DRAM- NFV algorithm reduces the scheduling rate for the corresponding service class queue at the egress router of the upstream domain in the event of congestion. This process will reduce the number of packets which are awaiting service in the corresponding service class queue at the ingress router of the downstream domain and thus, the average queue delay is also reduced. Figure 6-15, Figure 6-16 and Figure 6-17 illustrate the average queue delay at the ingress router of the downstream domain in relation to the DWFQ and DRAM-NFV algorithms. Moreover, the DRAM-NFV algorithm reallocate the resources of the link that connects the DiffServ domains, after reducing the scheduling rates, such that less resources are allocated at the egress router of the upstream domain for the congested service class queues and more resources are dedicated at the egress router of the upstream domain for the uncongested service class queues. This process of dedicating extra resources allows some extra traffic to be forwarded from the uncongested class at the egress router of the upstream domain towards the ingress router of the downstream domain.

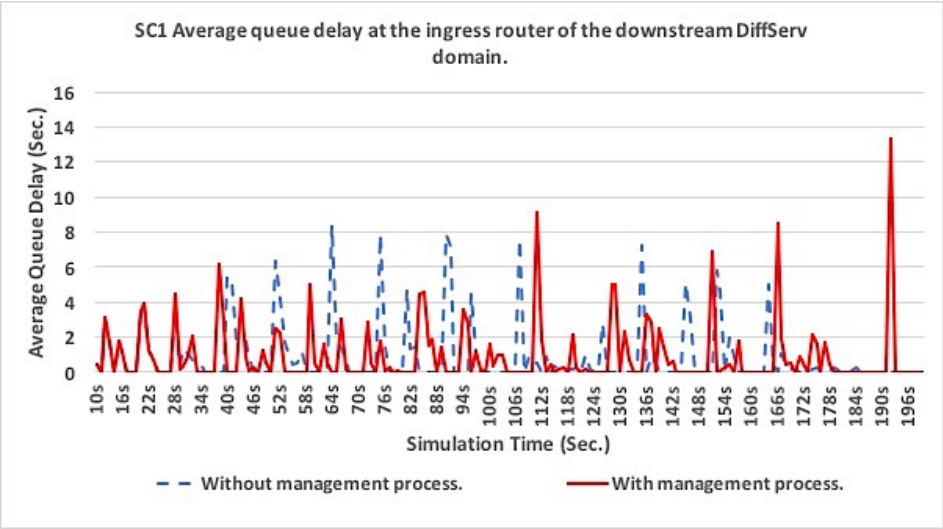


Figure 6-15, SC1 average queue delay - case study 2, test scenario 1.

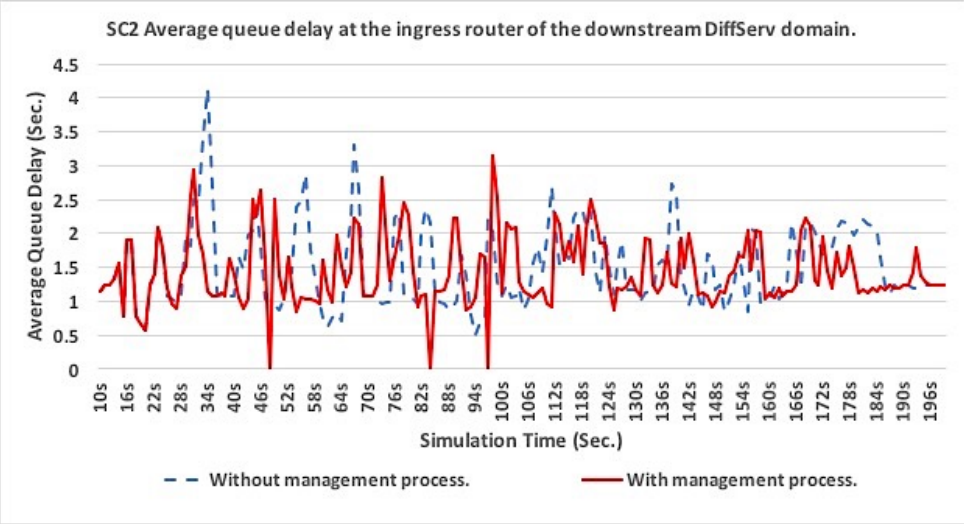


Figure 6-16, SC2 average queue delay - case study 2, test scenario 1

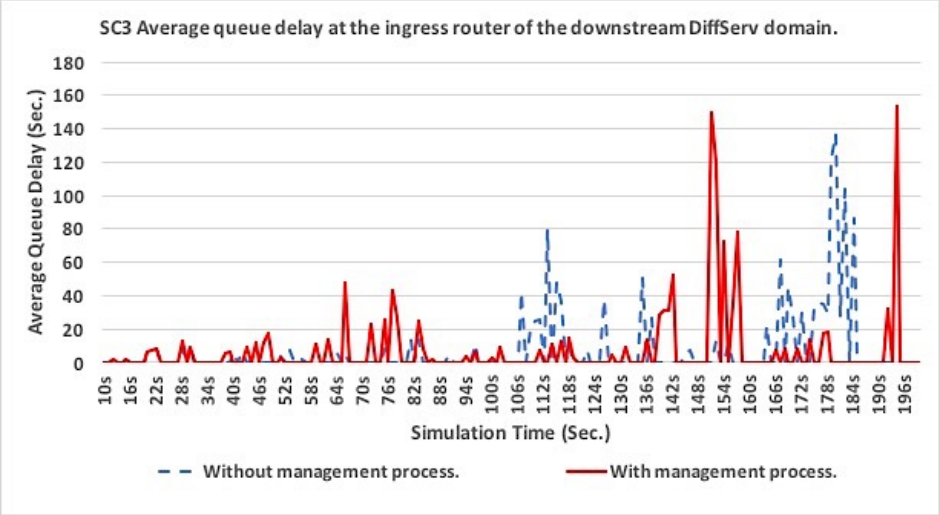


Figure 6-17, SC3 average queue delay - case study 2, test scenario 1

When the traffic of the medium and lowest priority service classes (SC_2 and SC_3) are increased, in case study 2, the utilization of the link that connects the DiffServ domains for SC_2 and SC_3 traffic is reduced and is organised according to the traffic conditions in the ingress router of the downstream domain and the traffic conditions in the egress router of the upstream domain. Table 6-8 illustrates the average percentage utilizations for the service classes traffic at the link that connects the upstream and downstream domains for the whole period of simulation.

In terms of average End to End Delay, Table 6-9 illustrates the average End to End Delay for service classes traffic in the upstream and downstream DiffServ domains for the whole period of simulation.

Table 6-8, Average percentage utilization for the service classes at the link that connects DiffServ domains - Test scenario 1, case study 2.

Service Class	Domains Link utilization with DWFQ algorithm	Domains Link utilization with DRAM-NFV algorithm
SC1	4.692%	4.799%
SC2	40.422%	40.166%
SC3	1.208%	1.172%

Table 6-9, Average End-to-End Delay, test scenario 1 - case study 2.

Service Class	Average End to End Delay using DWFQ algorithm	Average End to End Delay using DRAM-NFV algorithm	Increased/Decreased
SC1	4.4226 sec	4.584 sec	Increased by 0.16 sec.
SC2	4.356 sec	4.454 sec	Increased by 0.1 sec.
SC3	84.353 sec	93.909 sec	Increased by 9.56 sec.
Overall traffic	10.937 sec	11.295 sec	Increased by 0.358 sec.

The use of DRAM-NFV causes some extra delays for the lowest priority service class (SC_3). This increment in SC_3 average End to End Delay does not have an effect on the QoS for the SC_3 queue because this class is always dedicated to the traffic that can sustain delays and packet drops. The average End to End Delay for SC_1 and SC_2 traffic is increased slightly. Although the delay of SC_1 traffic is increased slightly with the use of the DRAM-NFV algorithm, its traffic utilization across the DiffServ domains in the event of congestion is improved.

6.6.3 Case Study 3- Test Scenario 1 Analysis:

In the third case study for the test scenario 1- Figure 6-1, the traffic of all service classes is increased by creating more SC_1, SC_2 and SC_3 traffic sources at the DiffServ traffic source node (n_0) of the simulation test scenario (1) topology diagram , appendix A-5.1- Figure 9-8. First, as before the DWFQ algorithm is applied to the management of resources within the edge routers of the DiffServ domains. The service class queues at the ingress router of the downstream domain then suffer from different levels of congestion as shown in Figure 6-18, Figure 6-19 and Figure 6-20 (dashed curves).

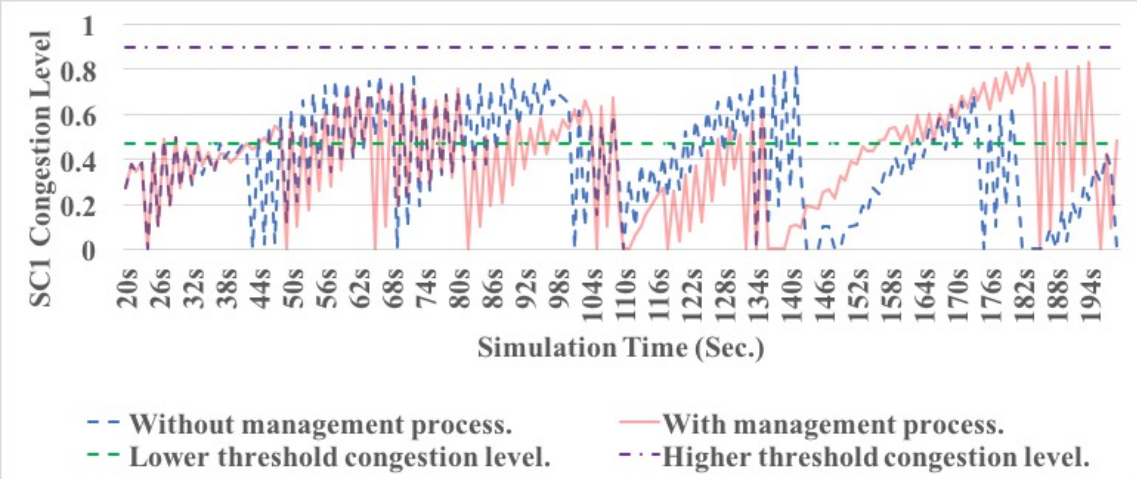


Figure 6-18, SC1 congestion level - case study 3, test scenario 1

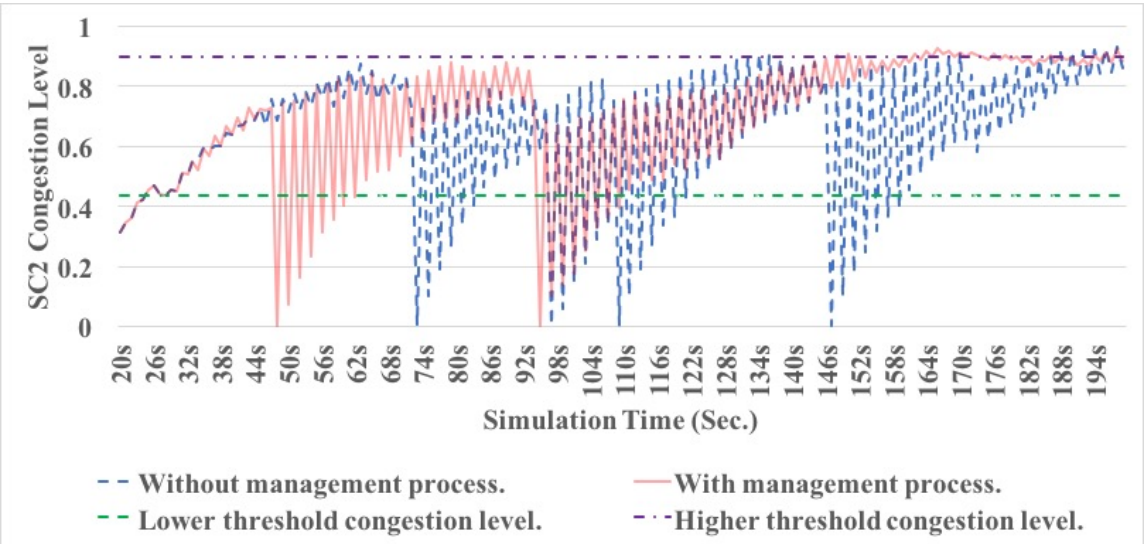


Figure 6-19, SC2 congestion level - case study 3, test scenario 1

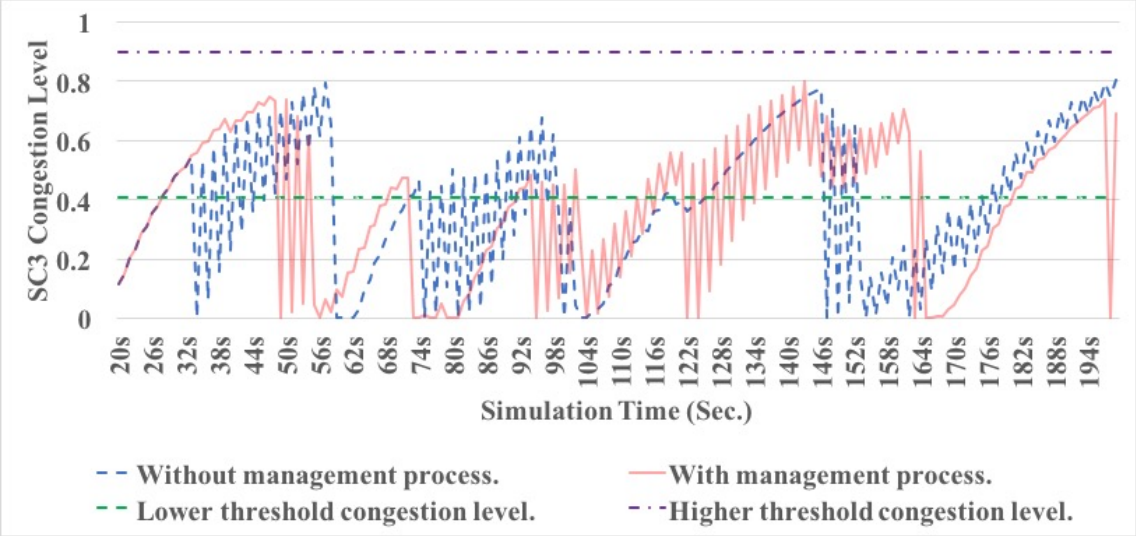


Figure 6-20, SC3 congestion level - case study 3, test scenario 1

When the DRAM-NFV algorithm is applied in place of the DWFQ algorithm, the utilizations of the link that connects the DiffServ domains for SC_1 and SC_2 traffic are increased while the SC_3 traffic utilization is decreased. The link's resources is organised according to the traffic conditions in the ingress router of the downstream domain and the traffic conditions in the egress router of the upstream domain. Table 6-10 illustrates the average percentage utilizations for the service classes traffic at the link that connects the upstream and downstream domains for the whole period of simulation. In DiffServ, it is impossible to improve the QoS for all service classes because all the service class queues use weights to share one out link in forwarding their traffic.

Table 6-10, Average percentage utilization for the service classes at the link that connects DiffServ domains - Test scenario 1, case study 3.

Service Class	Domains Link utilization with DWFQ algorithm	Domains Link utilization with DRAM-NFV algorithm
SC1	7.191%	7.251%
SC2	33.072%	33.167%
SC3	1.283%	1.091%

Although there is this improvement in the SC_1 and SC_2 traffic utilizations at the link that connects DiffServ domains which occurs when the DRAM-NFV algorithm is applied, the SC_1 utilization at the destination link is reduced because of the best effort traffic that is injected to change the network traffic conditions of the downstream domain through changing the traffic conditions in the egress router for the downstream domain. The SC_3 traffic utilization at the

destination link is also reduced while the SC_2 traffic utilization is increased. Table 6-11 illustrates the average percentage service classes' traffic utilizations at the destination for the whole period of simulation.

Table 6-11, Average percentage utilization for the service classes at the destination link - Test scenario 1, case study 3.

Service Class	Destination Link utilization with DWFQ algorithm	Destination Link utilization with DRAM-NFV algorithm
SC1	1.80%	1.75%
SC2	14.575%	14.9%
SC3	0.62%	0.55 %

In terms of average End to End Delay, the DRAM-NFV algorithm participates in reducing the average End to End Delay for overall traffic. The average End to End Delay for the highest and medium priority service classes are kept approximately the same. However, the average End to End Delay for the lowest priority service class SC_3 is decreased. Table 6-12 illustrates the average End to End Delay for service classes traffic in the upstream and downstream DiffServ domains for the whole period of simulation.

Table 6-12, Average End-to-End Delay, test scenario 1 - case study 3.

Service Class	Average End-to-End delay using DWFQ algorithm	Average End-to-End delay using DRAM-NFV algorithm	Increased/ Decreased
SC1	4 sec	4 sec	-
SC2	4.6 sec	4.6 sec	-
SC3	107.388 sec	103.715 sec	Decreased by 3.67 sec
Overall traffic	11.214 sec	10.281	Decreased by 0.93 sec

6.7 Test Scenario 2 Analysis:

In test scenario 2, the upstream domain consists of a single core router while the downstream domain consists of a number of multiple core routers, as shown in Figure 6-2. The simulation diagram of the test scenario 2 topology is illustrated in appendix A-5.2- Figure 9-9. The configuration parameters of test scenario 2 is shown in Table 6-13:

In the downstream domain, the packets are routed randomly within the domain. Best effort traffic (SC_4) is injected into the downstream domain through the core routers to change the traffic conditions within that domain. This traffic can be injected into any of the service class queues, depending on the available resources in those queues. The edge routers in the upstream and downstream domains use dynamic weights to forward their traffic while the core routers use pre-configured fixed weights; they forward packets from their queues based on this sequence (SC_1:SC_2:SC_3) (5:4:3) packets respectively as justified that in 6.6.

Each out-port of the ingress router for the downstream domain contains a queue for each defined service class. The scheduling rate for each class is configured by taking the maximum average service class queue length and the minimum average queue delay to maximize the throughput of service classes, as explained in section 4.4. The service class congestion level at these out-port queues is measured by taking the maximum average service class queue length at these out-port queues, as explained previously in section 4.4.

In order to test the performance of the DRAM-NFV algorithm in managing resources across the upstream and downstream domains of test scenario 2, first, the traffic of a selected service class is increased individually. This increment in the service class traffic causes congestion in the selected service class queue and also in some other queues because all service classes queues in DiffServ are competing between themselves to occupy the out link capacity of the DiffServ router. The performance of the DRAM-NFV algorithm is then compared with the performance of the DWFQ algorithm – this latter does not manage resources across DiffServ domains.

Table 6-13, Configuration parameters of test scenario 2.

Parameter	Configuration value	Notes
Link capacities of the upstream domain	5 Mbits	To ensure that more DiffServ traffic is injected into the downstream domain from the upstream domain.
Link capacity that connects the upstream and downstream domains.	10 Mbits	
link capacities of the downstream domain	1 Mbits	
Link capacity that connects the downstream domain and destination.	3 Mbits	
Both the upstream and downstream domains provide three service class queues	(SC_1, SC_2, SC_3)	
The delay configuration parameters (δ) for service classes (SC_1:SC_2:SC_3) at the edge DiffServ routers.	(1:2:4)	(Chin-Chang Li et al. 2000).
The service class markings in the upstream and downstream domains.	Identical.	
Service class queue buffer size in the upstream and downstream domains.	128 packets	(Wendell Odom 2005).
Minimum (lower) Service class queue threshold values (SC_1:SC_2:SC_3) in the upstream and downstream domains.	(15/32:7/16:13/32) times the queue size.	(Cisco 2013)
Service class queue dropping probabilities (SC_1:SC_2:SC_3) in the upstream and downstream domains.	(1/2:1/2:1/2) times the queue size.	(Cisco 2013)
Traffic Distribution in the upstream and downstream domains.	VoIP (SC1) – Highest Priority, Video Traffic (SC2), FTP (SC3) – Lowest Priority, and Data Base Request message (Best Effort).	
Lower threshold congestion values for (SC_1:SC_2:SC_3) in the downstream domain	(15/32:7/16:13/32)	(Cisco 2013)
Higher threshold congestion values for (SC_1:SC_2:SC_3) in the downstream domain	(0.9:0.9:0.9)	To get a linear scale of resources management
Maximum Scheduling rate update factor (β_{max}) for (SC_1:SC_2:SC_3) in the downstream domain.	(0.9:0.9:0.9)	To damp the throughput at upstream domain.

In test scenario (2), two case studies are undertaken to test the performance of the DRAM-NFV algorithm. These are as follows:

6.7.1 Case Study 1- Test Scenario 2 Analysis:

In case study 1 of test scenario 2 - Figure 6-2, only the traffic of the lowest priority service class queue (SC_3) increases through increasing the number of (SC_3) sources that are created at the node (n_0) of the test scenario 2 topology, appendix A-5.2- Figure 9-9. The service class queues at the out ports of the ingress router for the downstream domain ($Q_{268,}$ Q_{269} and Q_{2610}) also suffer from levels of congestion. These levels are measured for the circumstance in which the DWFQ algorithm only is used for scheduling, as illustrated in Figure 6-21, Figure 6-22 and Figure 6-23 (dashed curves).

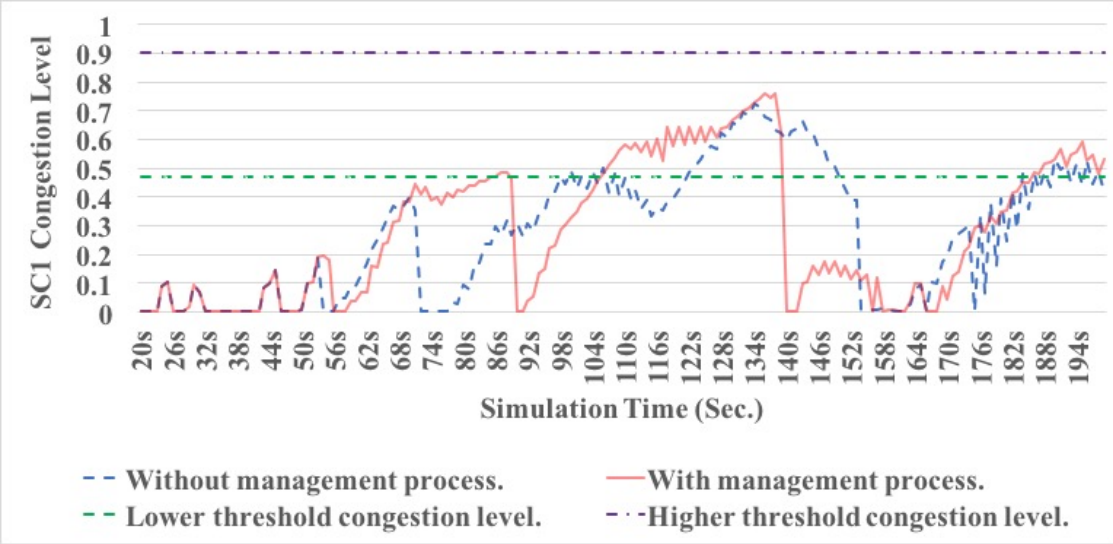


Figure 6-21,SC1 maximum congestion level using the DWFQ and DRAM-NFV algorithms – case study 1, test scenario 2

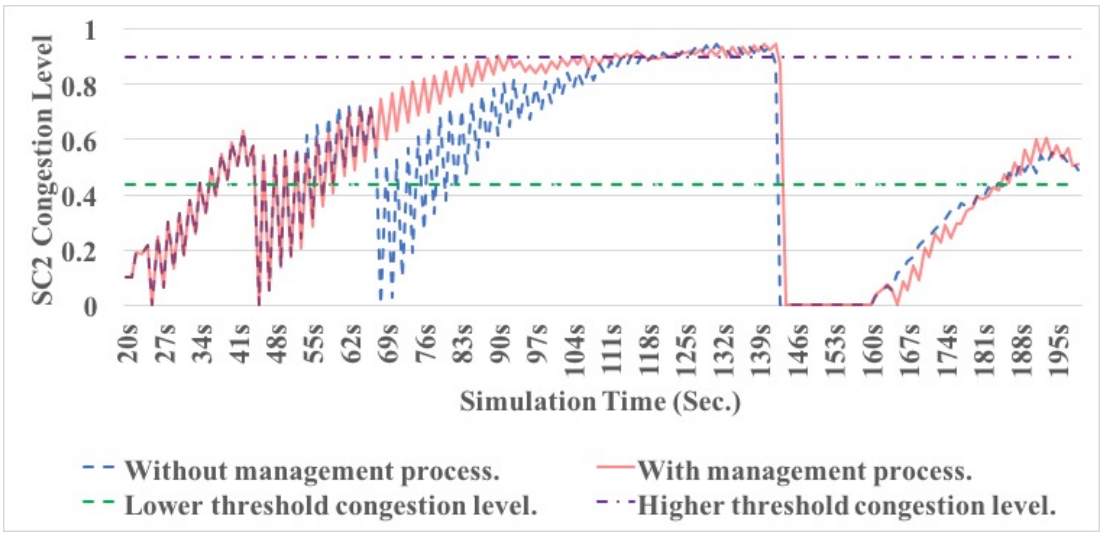


Figure 6-22, SC2 maximum congestion level using the DWFQ and DRAM-NFV algorithms - case study 1, test scenario 2

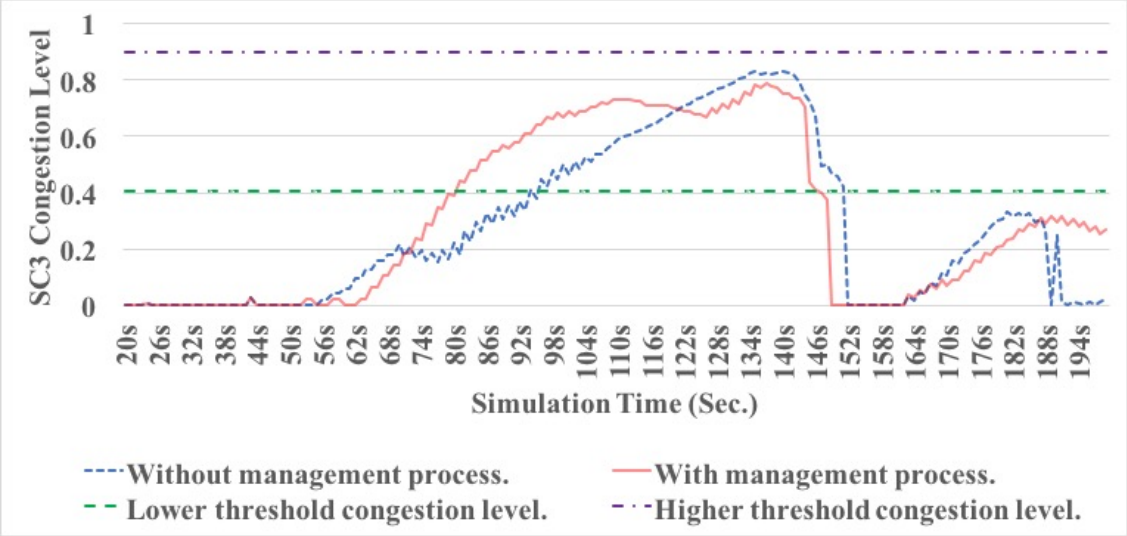


Figure 6-23, SC3 maximum congestion level using the DWFQ and DRAM-NFV algorithms - case study 1, test scenario 2

When managing the resources of the upstream and downstream domains using the DRAM-NFV algorithm, the utilization of the link that connects the DiffServ domains is organised based on the traffic conditions in the out port queues of the ingress router of the downstream domain and the traffic conditions in the egress router of the upstream domain.

By comparing the utilization of that link when using the DWFQ and then the DRAM-NFV algorithms in turn, it can be seen that the DRAM-NFV algorithm prevents the highest priority service class (SC_1) from occupying the required bandwidth for its traffic at the egress router of the upstream domain whilst congestion is occurring. This means that the DRAM-NFV algorithm grants some extra resources to SC_2 traffic at the egress router of the upstream domain at the expense of SC_1 and the increasing SC_3 traffic. This extra resource causes an increment in the SC_2 congestion level at the out ports queues for the ingress router, as shown in Figure 6-22. Table 6-14 illustrates the average percentage utilizations for the service classes traffic at the link that connects the upstream and downstream domains for the whole period of simulation.

Table 6-14, Average percentage utilization for the service classes at the link that connects DiffServ domains - Test scenario 2, case study 1.

Service Class	Domains Link utilization with DWFQ algorithm	Domains Link utilization with DRAM-NFV algorithm
SC1	2.241%	2.09%
SC2	23.557%	24.215%
SC3	0.371%	0.298%

The average End to End Delay for all the service class traffic in the upstream and downstream DiffServ domains for the whole period of simulation is shown in Table 6-15. The average End to End Delays for SC_1 and SC_2 traffic are reduced slightly while the average End to End Delay for the lowest priority service class (SC_3) is reduced dramatically.

Table 6-15, Average End-to-End Delay, test scenario 2 - case study 1.

Service Class	Average End to End Delay using DWFQ algorithm	Average End to End Delay using DRAM-NFV algorithm	Increased/Decreased
SC1	5.98 sec	5.85 sec	Decreased by 0.13 sec.
SC2	5.51 sec	5.46 sec	Decreased by 0.05 sec.
SC3	123.6 sec	105.74 sec	Decreased by 17.86 sec.
Overall traffic	11.3 sec	10.6 sec	Decreased by 0.7 sec.

6.7.2 Case Study 2- Test Scenario 2 Analysis:

In case study 2 of test scenario 2 - Figure 6-2, the traffic of all the service class queues is increased by increasing the number of service class traffic source applications at the DiffServ traffic sources node (n_0) of the simulation scenario (2) topology diagram, appendix A-5.2-Figure 9-9. The congestion levels (α) for the service class (SC_1, SC_2 and SC_3) queues at the out-ports of the ingress router (Q_{268}, Q_{269} and Q_{2610}) of the downstream domain are then measured. This is carried out, first, in relation to the use of the DWFQ algorithm only which allocates resources within the edge routers of a DiffServ domain only. Figure 6-24, Figure 6-25 and Figure 6-26 (dashed curves) illustrate the measured maximum congestion levels in the queues of the service classes (SC_1, SC_2 and SC_3) at the out-ports for the ingress router of the downstream domain. It is noticeable that all service class queues at these out-ports suffer from various levels of congestions when the DWFQ algorithm is applied. However, the (SC_2) queue is considerably more congested than the other classes.

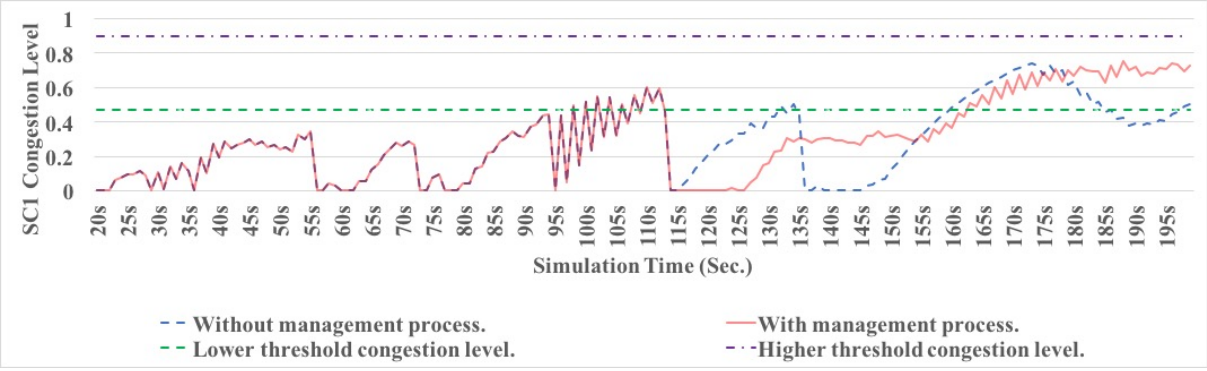


Figure 6-24, SC1 maximum congestion level using the DWFQ and DRAM-NFV algorithms – case study 2, test scenario 2

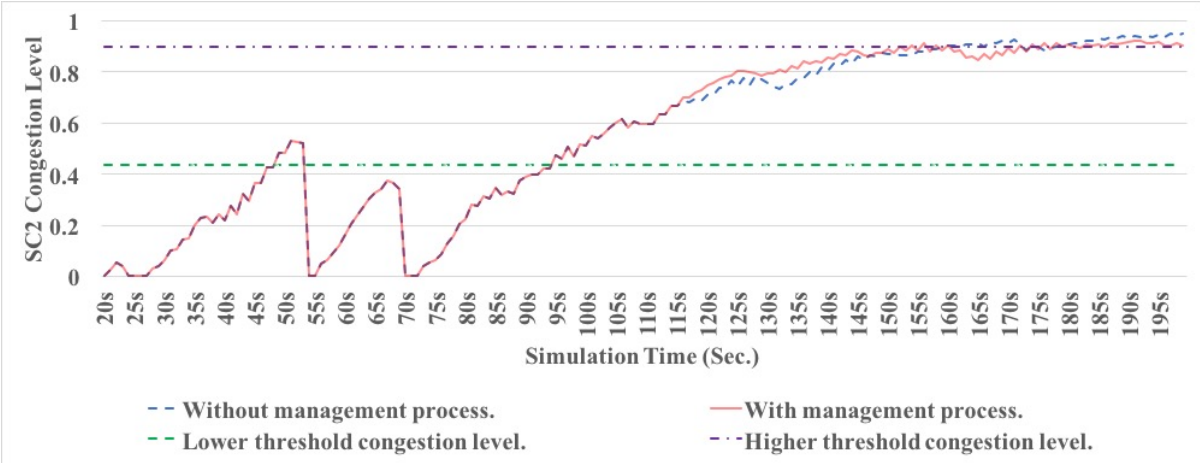


Figure 6-25, SC2 maximum congestion level using the DWFQ and DRAM-NFV algorithms – case study 2, test scenario 2.

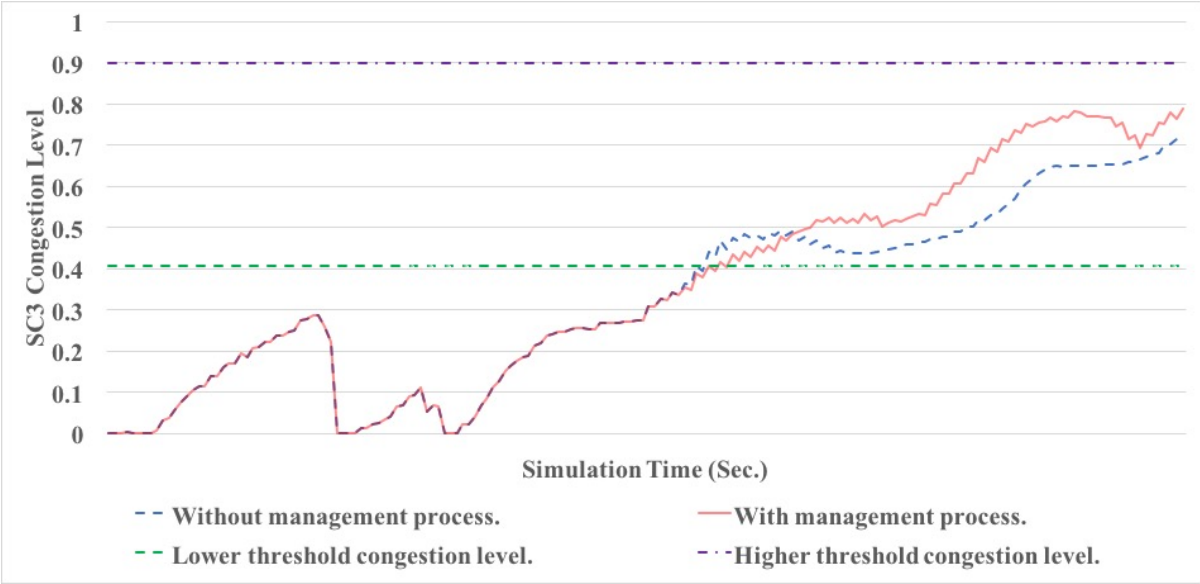


Figure 6-26, SC3 maximum congestion level using the DWFQ and DRAM-NFV algorithms – case study 2, test scenario 2.

When the DRAM-NFV algorithm is applied to manage resources across the different DiffServ domains (and within the edge routers of DiffServ domains), in the event of congestion, the utilization of the link that connects the upstream and downstream domains for SC_1 and SC_3 traffic is increased, whilst congestion occurs in the SC_2 queues because the DRAM-NFV algorithm works by reducing the scheduling rates of the congested service class queue at the egress router of the upstream domain and reallocates the resources of the link that connects the DiffServ domains. Table 6-16 illustrates the average percentage utilizations for the service class traffic at the link that connects the upstream and downstream domains for the whole period of simulation.

Table 6-16, Average percentage utilization for the service classes at the link that connects DiffServ domains - Test scenario 2, case study 2.

Service Class	Domains Link utilization with DWFQ algorithm	Domains Link utilization with DRAM-NFV algorithm
SC1	2.2%	2.25%
SC2	25.1%	24.7%
SC3	0.335%	0.458%

In addition to improving the utilization of the link that connects the DiffServ domains, the utilizations at the destination of the SC_1 and SC_3 traffic are also increased while the SC_2 traffic utilization at that destination is decreased. Table 6-17 illustrates the average percentage utilizations for the service classes traffic at the destination link for the whole period of simulation.

Table 6-17, Average percentage utilization for the service classes at the destination link - Test scenario 2, case study 2.

Service Class	Destination Link utilization with DWFQ algorithm	Destination Link utilization with DRAM-NFV algorithm
SC1	1.358%	1.762%
SC2	23.606%	20.7783%
SC3	0.479%	0.8325%

when resources are managed using the DRAM-NFV algorithm, the average End to End Delay for the highest priority service class traffic (SC_1) is decreased while the average End to End Delay for the congested service class queue (SC_2) and for the lowest priority service class traffic (SC_3) are increased. Table 6-18 shows the average End-to-End Delay of service classes traffic in the upstream and downstream domains for the whole period of simulation.

Table 6-18, Average End-to-End Delay, test scenario 2 - case study 2.

Service Class	Average End to End Delay using DWFQ algorithm	Average End to End Delay using DRAM-NFV algorithm	Increased/Decreased
SC1	6.7 sec	6.4 sec	Decreased by 0.3 sec.
SC2	5.47 sec	5.8 sec	Increased by 0.33 sec.
SC3	85.097 sec	116.07 sec	Increased by 31 sec.
Overall traffic	7.79 sec	11.99 sec	Increased by 4.2 sec.

The DRAM-NFV algorithm improves the utilization of the highest priority service class traffic (SC_1) and reduces its average End to End Delay throughout the time that the (SC_2) queue is congested. Moreover, it improves the utilization of the lowest priority service class (SC_3) whereas it causes some extra delay to its traffic. This increment in SC_3 average End to End Delay does not effect the QoS for the (SC_3) queue because this class is dedicated to the traffic that can most sustain delays and packet drops, while reducing the End to End Delay and increasing the utilization (throughput) of (SC_1) are considered important factors in regard to improving the QoS of (SC_1) when congestion occurs in SC_2 the queue because the traffic of SC_1, which is VoIP traffic as detailed in section 6.2, is very sensitive to delay and packet drop.

6.8 Test Scenario 3 Analysis:

In test scenario 3, there are two upstream DiffServ domains that forward their traffic to one downstream DiffServ domain, as shown in Figure 6-3. One of the upstream domains (Domain 1) is configured to attempt to manage resources across the domains while the other domain (Domain 3) only manages its own egress router resources and does not take into consideration the traffic conditions in the downstream domain. The simulation diagram of the test scenario 3 topology is illustrated in appendix A-5.3, Figure 9-10. The upstream domains consist of single core routers while the downstream domain consists of two ingress routers with multiple out ports connected to a multi-core router arrangement which sends packets to one egress router with a single out port, as illustrated in Figure 6-3. The configuration parameters of test scenario 3 is shown in Table 6-19.

In the downstream domain, packets are routed randomly within the domain. Best effort traffic (SC_4) is not injected into the downstream domain. The edge routers in the upstream and downstream domains use dynamic weights to forward their traffic while the core routers use pre-configured fixed weights; they forward packets from their queues based on this sequence (SC_1:SC_2:SC_3) (5:4:3) packets respectively as justified that in 6.6.

The scheduling rates and the congestion levels of each service class queue at each out-port of the ingress routers for the downstream domain are determined by taking the maximum average queue length and minimum average queue delay at these out ports, as explained in section 4.4. The service class scheduling rate at all the out-ports is the same and the process of calculating the congestion level in a specific service class is based on the maximum average service class queue length across all the out-port queues of the ingress routers.

Three case studies are examined in order to evaluate the performance of the DRAM-NFV algorithm in managing resources across the upstream and downstream domains of test scenario

3 whilst congestion is occurring. This performance is compared with that of the DWFQ algorithm which does not manage resources across different DiffServ domains. These cases are as follows:

Table 6-19, Configuration parameters of test scenario 3.

Parameter	Configuration value	Notes
Link capacities of the upstream domain	5 Mbits	To ensure that more DiffServ traffic is injected into the downstream domain from the upstream domain.
Link capacity that connects the upstream and downstream domains.	6 Mbits	
link capacities of the downstream domain	1 Mbits	
Link capacity that connects the downstream domain and destination.	3 Mbits	
Both the upstream and downstream domains provide three service class queues	(SC_1, SC_2, SC_3)	
The delay configuration parameters (δ) for service classes (SC_1:SC_2:SC_3) at the edge DiffServ routers.	(1:2:4)	(Chin-Chang Li et al. 2000).
The service class markings in the upstream and downstream domains.	Identical.	
Service class queue buffer size in the upstream and downstream domains.	128 packets	(Wendell Odom 2005).
Minimum (lower) Service class queue threshold values (SC_1:SC_2:SC_3) in the upstream and downstream domains.	(15/32:7/16:13/32) times the queue size.	(Cisco 2013)
Service class queue dropping probabilities (SC_1:SC_2:SC_3) in the upstream and downstream domains.	(1/2:1/2:1/2) times the queue size.	(Cisco 2013)
Traffic Distribution in the upstream and downstream domains.	VoIP (SC1) – Highest Priority, Video Traffic (SC2), FTP (SC3) – Lowest Priority, and Data Base Request message (Best Effort).	
Lower threshold congestion values for (SC_1:SC_2:SC_3) in the downstream domain	(15/32:7/16:13/32)	(Cisco 2013)
Higher threshold congestion values for (SC_1:SC_2:SC_3) in the downstream domain	(0.9:0.9:0.9)	To get a linear scale of resources management
Maximum Scheduling rate update factor (β_{max}) for (SC_1:SC_2:SC_3) in the downstream domain.	(0.9:0.9:0.9)	To damp the throughput at upstream domain.

6.8.1 Case Study 1- Test Scenario 3 Analysis:

In case study 1 of test scenario 3 - Figure 6-3, the traffic of the medium priority service class (SC_2) at the upstream domain (Domain 3) is only increased such that the numbers of the service class traffic sources at (n_0) , appendix A-5.3 - Figure 9-10, of the upstream domain (Domain 1) are (90:90:90) sources for (SC_1:SC_2:SC_3) respectively and the numbers of service class traffic sources at (n_1) of (Domain 3) are (90:120:90) sources for (SC_1:SC_2:SC_3) respectively. The maximum congestion level of the queues that belong to a specific service class at the out ports of the ingress routers for the downstream domain is measured, first, using the DWFQ algorithm. Figure 6-27, Figure 6-28 and Figure 6-29 show comparisons of maximum congestion levels in service class queues when using the DRAM-NFV algorithm as compared to the situation which pertains when using DWFQ algorithm only. It is noticeable that the (SC_1) and (SC_2) queues suffer from different levels of congestion with DWFQ algorithm, Figure 6-27 and Figure 6-28 (dashed curves). The SC_2 queues at the ingress routers are considerably more congested than the (SC_1) queues while the (SC_3) queues are the least congested queues, Figure 6-29 (dashed curves).

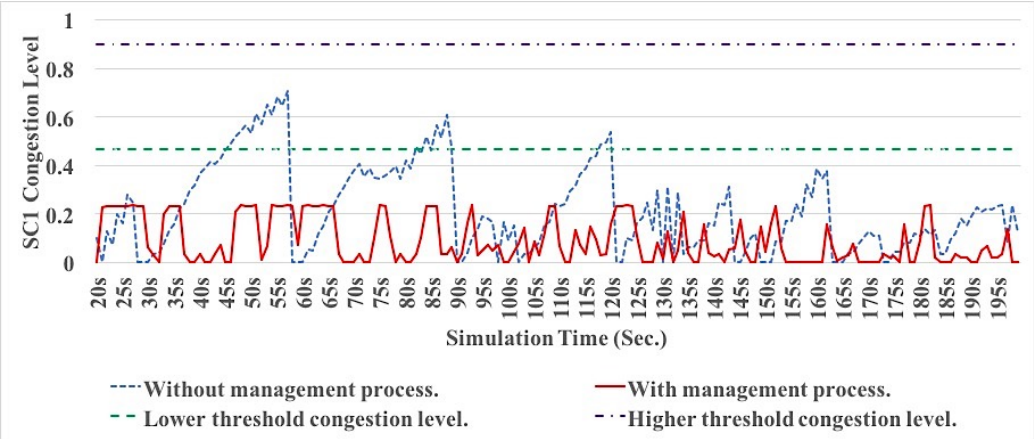


Figure 6-27 SC1 maximum congestion levels using the DWFQ and DRAM-NFV algorithms - case study 1, test scenario 3.

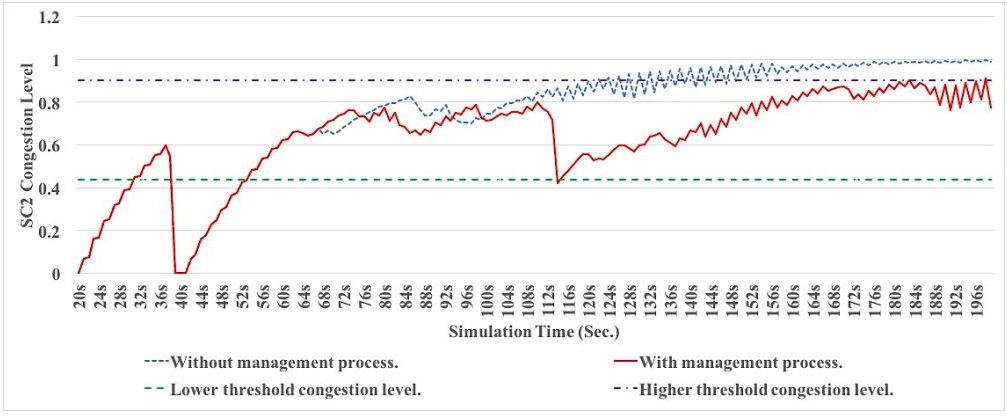


Figure 6-28, SC2 maximum congestion levels using the DWFQ and DRAM-NFV algorithms – case study 1, test scenario 3.

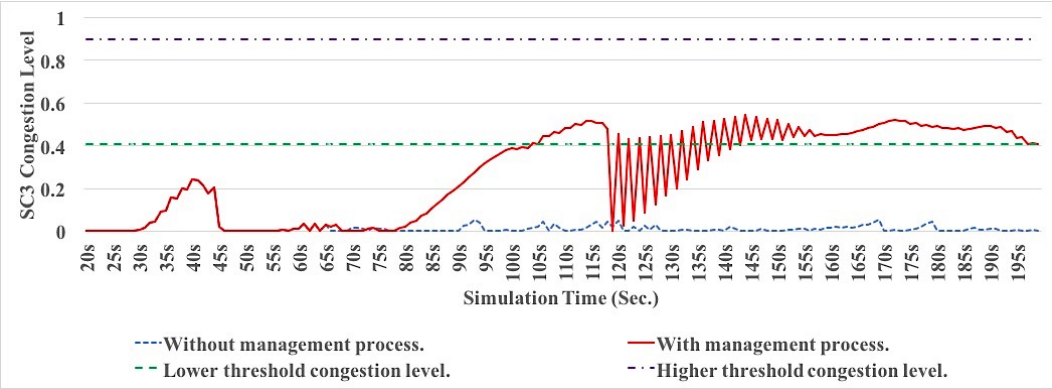


Figure 6-29, SC3 maximum congestion levels using the DWFQ and DRAM-NFV algorithms - case study 1, test scenario 3.

When the DRAM-NFV algorithm is applied, the congestion level in a service class queue can be reduced throughout periods of congestion but are increased during periods in which there is no congestion. As has been stated previously, the algorithm reduces the scheduling rate of the equivalent service class queue at the egress router of the upstream domain and reallocates the resource of the link that connects the upstream and downstream domains after that, thus this will reduce the number of packets that are waiting for service in the congested class and increase the number of packets in the uncongested class at the ingress router of the downstream domain. When the average service class queue length is reduced, the average service class queue delay is also reduced because the average service class queue delay is directly proportional to the average service class queue length.

Through the use of the DRAM-NFV algorithm, the resources of the link that connects the upstream domain (Domain 1) with the downstream domain, see Figure 6-3 – are reallocated since the service classes' scheduling rates take into account the traffic conditions in the ingress routers of the downstream domain and the traffic conditions at the egress router of the upstream domain (Domain 1). The utilizations of some service classes' traffic at the abovementioned link are improved such that the highest and the lowest priority service classes (SC_1) and (SC_3) traffic utilizations are increased throughout the presence of congestion in the (SC_2) queue at the ingress routers of the downstream domain. In contrast, the average percentage utilization for the medium priority service class (SC_2), which is considerably more congested than the other queues, is reduced. Table 6-20 illustrates the average percentage utilizations for the service classes traffic at the link that connects the upstream (Domain 1) and downstream domains for the whole period of simulation.

Table 6-20, Average percentage utilization for the service classes at the link that connects the upstream (Domain 1) and downstream DiffServ domains - Test scenario 3, case study 1.

Service Class	Domains Link utilization with DWFQ algorithm	Domains Link utilization with DRAM-NFV algorithm
SC1	6.706%	8.223%
SC2	44.674 %	40.437%
SC3	0.47%	0.52 %

It can be seen that the utilization of the (SC_1) traffic at the link that connects the upstream domain (Domain 1) to the downstream domain is improved. However, (SC_1) utilization at the destination link of the downstream domain is reduced because of the increasing (SC_2) traffic which is forwarded from the unmanaged upstream domain (Domain 3) towards the downstream domain. In contrast, the utilizations for the SC_2 and SC_3 traffic at the destination link increase. This encourages the study of the circumstance whereby the resources of all upstream domains are managed (across domains) in preference to the situation where a single upstream domain is managed in this way, while other upstream domains are not. Table 6-21 illustrates the average percentage utilizations for the service classes traffic at the destination link when manging the upstream domain (Domain 1) only for the whole period of simulation.

Table 6-21, Average percentage utilization for the service classes at the destination link when manging the upstream domain (Domain 1) only - Test scenario 3, case study 1.

Service Class	Destination Link utilization with DWFQ algorithm	Destination Link utilization with DRAM-NFV algorithm
SC1	3.379%	2.359%
SC2	15.22%	19.646 %
SC3	0.319%	0.35%

In terms of service classes' average End to End Delays, the use of the DRAM-NFV algorithm increases the average End to End Delay for SC_1 traffic while the average End to End Delays for SC_2 and SC_3 traffic are reduced**.** Table 6-22 shows the average End to End Delay for all the service class traffic in the upstream and downstream DiffServ domains for the whole period of simulation.

Table 6-22, Average End-to-End Delay when manging the upstream domain (Domain 1) only, test scenario 3 - case study 1.

Service Class	Average End to End Delay using DWFQ algorithm	Average End to End Delay using DRAM-NFV algorithm	Increased/Decreased
SC1	4.196 sec	5.556 sec	Increased by 1.36 sec.
SC2	8.263 sec	7.206 sec	Decreased by 1.057 sec.
SC3	120.388 sec	107.535 sec	Decreased by 12.853 sec.
Overall traffic	7.8 sec	8.25 sec	Increased by 0.45 sec.

On applying the DRAM-NFV algorithm to this scenario, the throughput of the highest priority service class (SC_1) traffic at the destination link of the downstream domain is decreased and its average End to End Delay increases whilst there is congestion in the SC_2 queues. This does damage the QoS for the highest priority service class traffic – which is highly sensitive to delay and packets drop.

In the second part of this case study, both the upstream domains (Domain 1) and (Domain 3) are managed based on the service class congestion levels at the ingress routers of the downstream domain. The congestion levels in the service class queues tend to be reduced during periods of congestion and increased during periods that when there is no congestion (relative to the situation where the DWFQ and DRAM-NFV algorithms are in use separately), as shown in Figure 6-30, Figure 6-31 and Figure 6-32. It is noticeable that the SC_1, SC_2 and SC_3 queues suffer from different levels of congestion, thus their scheduling rates at the egress routers of the upstream domains (Domain 1 and Domain 3 of Figure 6-3) are reduced based on the determined values for the scheduling rates update factor (β).

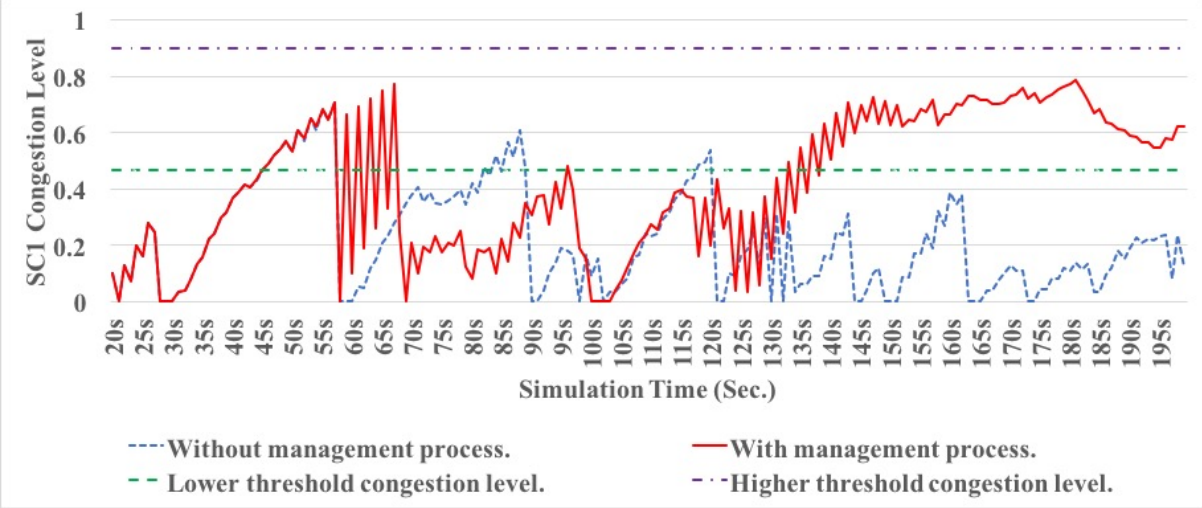


Figure 6-30, SC1 maximum congestion level when using a mix of the DWFQ and DRAM-NFV algorithms to manage the upstream domains - case study 1, test scenario 3.

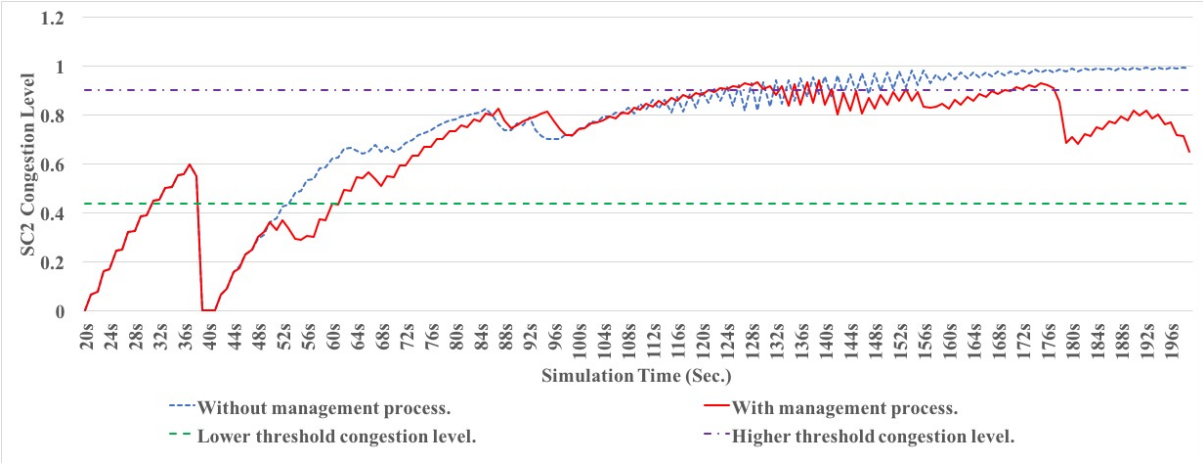


Figure 6-31, SC2 maximum congestion levels when using DWFQ and DRAM-NFV algorithms to manage both upstream domains - case study 1, test scenario 3.

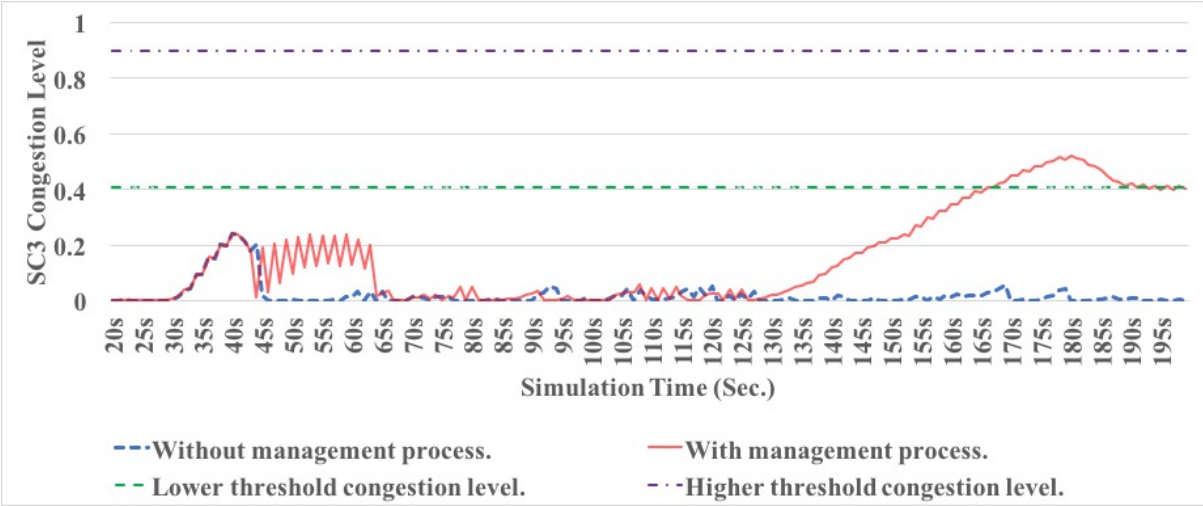


Figure 6-32, SC3 maximum congestion level using DWFQ and DRAM-NFV algorithms with managing both upstream domains - Study case 1, test scenario 3.

The SC_1 traffic utilizations at the (Domain 1- downstream) and (Domain 3- downstream) links are increased while The SC_2 traffic utilizations are reduced. The SC_3 traffic utilizations at these links stay approximately the same. Table 6-23 summarizes the service classes utilizations when both the upstream domain out links are managed by DRAM-NFV.

Table 6-23, Service classes utilizations when managing both the upstream domains out links - case study 1, test scenario 3.

Upstream domain out link (Domain 1- Downstream)	DWFQ Algorithm	DRAM-NFV Algorithm
SC1 Traffic	6.706%	7.632%
SC2 Traffic	44.674%	42.174%
SC3 Traffic	0.47%	0.445%

Upstream domain out link (Domain 3- Downstream)	DWFQ Algorithm	DRAM-NFV Algorithm
SC1 Traffic	7.99%	9.133%
SC2 Traffic	39.146%	36.038%
SC3 Traffic	0.406%	0.411%

The SC_1 traffic utilization at the destination link of the downstream domain is better than its utilization in the case of only one upstream domain (Domain 1) being managed. The SC_2 traffic utilization at that link is also increased but it is less than its utilization in the case where only (Domain 1) is managed. The SC_3 utilization is approximately the same in both cases. Table 6-24 summarizes the service classes' utilizations when managing both the upstream domains out links and when managing a single upstream domain out link (Domain 1).

Table 6-24, service classes utilizations at the destination link when managing both the upstream domains out links and when managing only a single upstream domain out link - case study 1, test scenario 3.

	When managing the out link of a single upstream domain (Domain 1)		When managing the out links of both upstream domains (Domain 1 and 3)	
Utilization at the Destination link.	DWFQ Algorithm	DRAM-NFV algorithm	DWFQ Algorithm	DRAM-NFV algorithm
SC1	3.379%	2.359%	3.379%	3.021%
SC2	15.22%	19.646%	15.22%	16.733%
SC3	0.319%	0.35%	0.319%	0.342%

In terms of service classes' average End to End Delays, the DRAM-NFV algorithm increases the average End to End Delay for the SC_1 traffic but this delay is less than that recorded when only one upstream domain (Domain 1) is managed. The average End to End Delays of the SC_2 and SC_3 traffic are reduced. Table 6-25 summarizes the service classes' average End to End Delays, first, when managing both the upstream domains out links and, second, when managing only a single upstream domain out link.

Table 6-25, service classes average End to End Delays when managing both the upstream domains out links and when managing only a single upstream domain out link - case study 1, test scenario 3.

	When managing the out link of both upstream domains (Domain 1 and 3)		**When managing the out link of a single upstream domain (Domain 1)**	
Average End to End Delay	**DWFQ Algorithm**	**DRAM-NFV algorithm**	**DWFQ Algorithm**	**DRAM-NFV algorithm**
SC1	4.196 sec	4.543 sec	4.196 sec	5.556 sec
SC2	8.263 sec	7.654 sec	8.263 sec	7.206 sec
SC3	120.388 sec	105.401 sec	120.388 sec	107.535 sec
Overall traffic	7.808 sec	7.865 sec	7.808 sec	8.251 sec

6.8.2 Case Study 2- Test Scenario 3 Analysis:

In case study 2 of test scenario 3 - Figure 6-3, the traffic of the lowest priority service class (SC_3) is increased such that the number of (SC_3) traffic sources is set to 180 at both the nodes (n_0) and (n_1), representing at the upstream domains - appendix A-5.3- Figure 9-10, while the numbers of SC_1 and SC_2 traffic sources are reduced to 60 sources at both (n_0) and (n_1). The maximum congestion level of the queues that belong to a specific service class at the out ports of the ingress routers for the downstream domain is measured, first, using the DWFQ algorithm. Figure 6-33, Figure 6-34 and Figure 6-35 show comparisons of maximum congestion levels in

service class queues when using the DRAM-NFV algorithm as compared to the situation which pertains when using the DWFQ algorithm. When using DWFQ algorithm, it is noticeable that all the classes suffer from congestion, at different levels. The SC_2 and SC_3 queues are considerably more congested than the SC_1 queue, Figure 6-33, Figure 6-34 and Figure 6-35 (dashed curves).

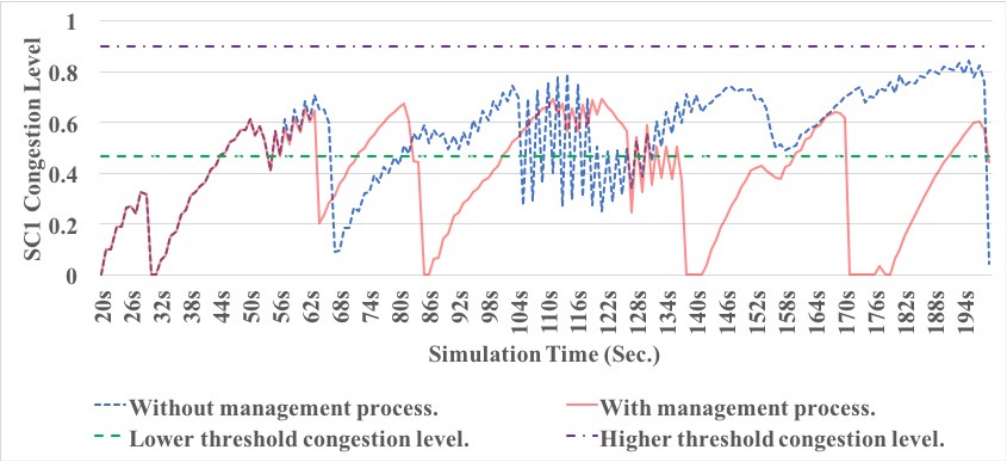


Figure 6-33, SC1 maximum congestion level using the DWFQ and DRAM-NFV algorithms - case study 2, test scenario 3.

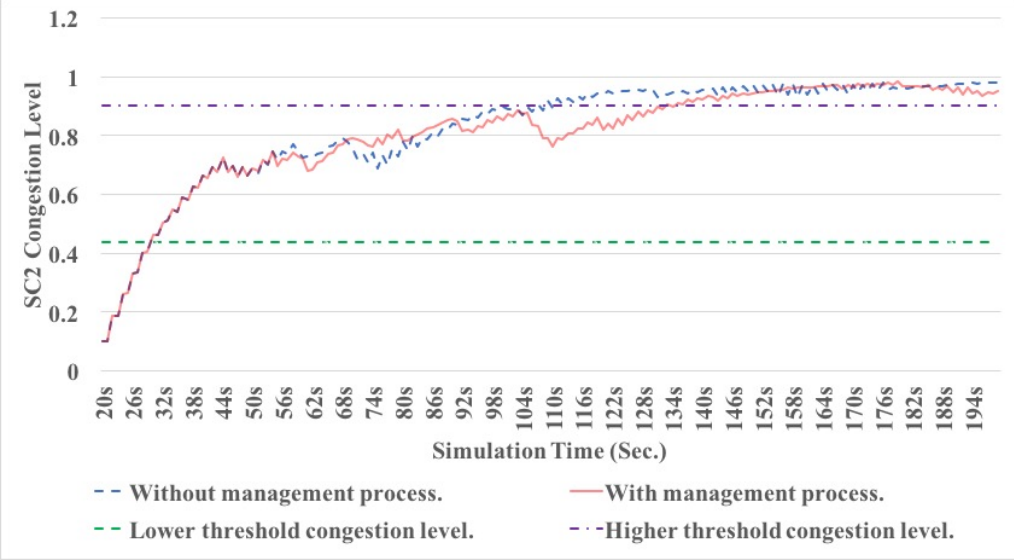


Figure 6-34, SC2 maximum congestion level using the DWFQ and DRAM-NFV algorithms - case study 2, test scenario 3.

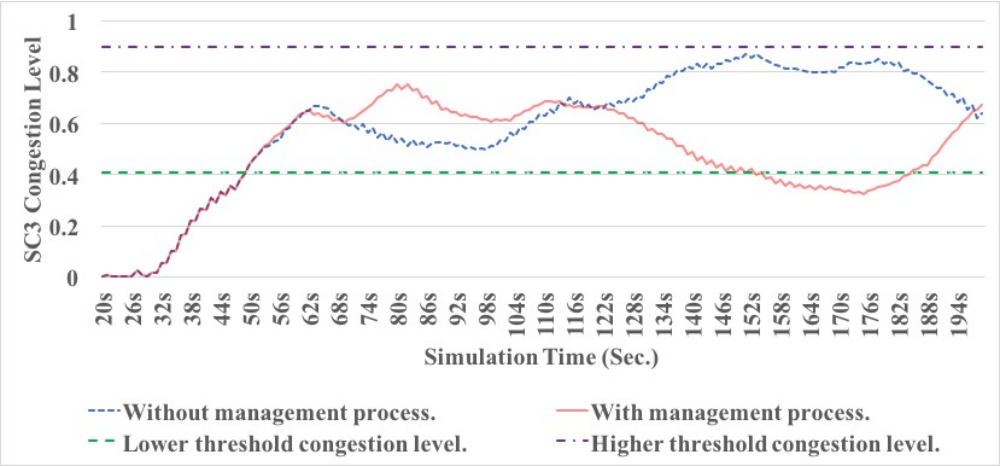


Figure 6-35, SC3 maximum congestion level using the DWFQ and DRAM-NFV algorithms - case study 2, test scenario 3.

When the DRAM-NFV algorithm is applied, the congestion level in the service class queues is reduced, generally, during periods of congestions and increased during periods that do not experience congestion, Figure 6-33, Figure 6-34 and Figure 6-35, as explained for the previous case studies. The congestion level represents the ratio of the average service class queue length to the service class buffer size so reducing the congestion level means reducing the average service class queue length. Thus, the average service class queue delay is also reduced because the average service class queue delay is directly proportion to the average service class queue length.

The utilization of the link that connects the upstream domain (Domain 1) with the downstream domain is also organized via the use of the DRAM-NFV algorithm. This organizational effort is based on the traffic conditions in the ingress routers of the downstream domain and the traffic conditions at the egress router of the upstream domain (Domain 1). The use of DRAM-NFV improves the highest priority service class (SC1) traffic utilization at that link when congestion occurs in the SC2 and SC3 queues at the ingress routers of the downstream domain while the average percentage utilizations for the medium and lowest priorities service class queues (SC_2) and (SC3) respectively, which are considerably more congested than the SC1 queue, are reduced. Table 6-26 summarizes the average percentage utilizations for the service classes traffic at the link that connects the upstream (Domain 1) and downstream domains for the whole period of simulation.

Table 6-26, Average percentage utilization for the service classes at the link that connects the upstream (Domain 1) and downstream DiffServ domains - Test scenario 3, case study 2.

Service Class	Domains Link utilization with DWFQ algorithm	Domains Link utilization with DRAM-NFV algorithm
SC1	4.181%	5.127%
SC2	50.493 %	48%
SC3	0.604%	0.536 %

Comparing the service classes utilizations of the (Domain 1- downstream) link with the service classes utilizations of the (Domain 3 – downstream) link when the DRAM-NFV algorithm is in use, it can be seen that the SC_1 and SC_2 traffic utilizations at the (Domain 1- downstream) link are better than the corresponding traffic utilizations at the (Domain 3 – downstream) link although both upstream domains have the same configurations as regards service class queues and there are the same numbers of service class traffic sources at (n_0) and (n_1). Table 6-27 summarizes the average percentage utilizations for the service classes traffic at the links that connect the upstream and downstream domains when managing the upstream (Domain 1) only using the DRAM-NFV algorithm for the whole period of simulation.

Table 6-27, Average percentage utilization for the service classes at the links that connect the upstream and downstream DiffServ domains - Test scenario 3, case study 2.

Upstream (Domain 1) – Downstream Link Utilization			Upstream (Domain 3) – Downstream Link Utilization	
Service Class	with DWFQ algorithm	with DRAM-NFV algorithm	with DWFQ algorithm	(Domain 3) – Downstream Link is managed using DWFQ algorithm while (Domain 1) – Downstream Link is managed using DRAM-NFV algorithm
SC1	4.181%	5.127%	4.476%	4.458%
SC2	50.493%	48%	47.175%	47.468%
SC3	0.604%	0.536%	0.691%	0.528%

The utilization of (SC_1) traffic at the destination link of the downstream domain is also improved with the use of the DRAM-NFV algorithm while the average utilizations for the (SC_2) and (SC_3) traffic are reduced. Table 6-28 illustrates the average percentage utilizations for the service classes traffic at the destination link for the whole period of simulation.

Table 6-28, Average percentage utilization for the service classes at the destination link when manging the upstream domain (Domain 1) only - Test scenario 3, case study 2.

Service Class	Destination Link utilization with DWFQ algorithm	Destination Link utilization with DRAM-NFV algorithm
SC1	1.277%	1.452%
SC2	23.88 %	23.5%
SC3	0.46%	0.336 %

In terms of service classes' average End to End Delays, the DRAM-NFV algorithm reduces the average End to End Delay for the SC_1 and SC_2 traffic slightly and causes a little extra delay for the SC_3 traffic as explained in Table 6-29 . The throughput and average End to End Delay for the highest priority service class (SC_1) are improved when the DRAM-NFV algorithm is applied to manage the link between the upstream (Domain 1) and downstream domains and when the SC_2 and SC_3 queues at the ingress routers of the downstream domain are suffering from congestion.

Table 6-29, Average End-to-End Delay when manging the upstream domain (Domain 1) only, test scenario 3 - case study 2.

Service Class	Average End to End Delay using DWFQ algorithm	Average End to End Delay using DRAM-NFV algorithm	Increased/Decreased
SC1	9.394 sec	9.045 sec	Decreased by 0.35 sec.
SC2	6.743 sec	6.644 sec	Decreased by 0.1 sec.
SC3	102.475 sec	102.977 sec	Increased by 0.5 sec.
Overall traffic	9.7 sec	8.9 sec	Decreased by 0.8 sec.

6.8.3 Case Study 3- Test Scenario 3 Analysis:

In case study 3 of test scenario 3 - Figure 6-3, the number of service class traffic sources that are created at the (DiffServ traffic sources) nodes (n_0) and (n_1) of the simulated test scenario 3 topology diagram, appendix A-5.3 - Figure 9-10, is 90 sources for each (SC_1, SC_2, SC_3). This creates some congestion states in the service class queues at the downstream domain. The maximum congestion level of the queues that belong to a specific service class at the out ports of the ingress routers for the downstream domain is measured using the DWFQ algorithm first. Figure 6-36, Figure 6-37 and Figure 6-38 illustrate comparisons of maximum congestion levels in service class queues when using the DRAM-NFV algorithm as compared to the situation which pertains when using the DWFQ algorithm. When using DWFQ algorithm, it is noticeable that the service classes suffer from different levels of congestion. The SC_2 queue is considerably more congested than the SC_1 and SC_3 queues, Figure 6-36, Figure 6-37 and Figure 6-38 (dashed curves).

When the DRAM-NFV algorithm is applied to manage resources across and within the upstream and downstream domains, the congestion level in the service class queues is reduced throughout the periods of congestion, but increases during the periods that do not experience congestion, as explained in the section 6.8.1.

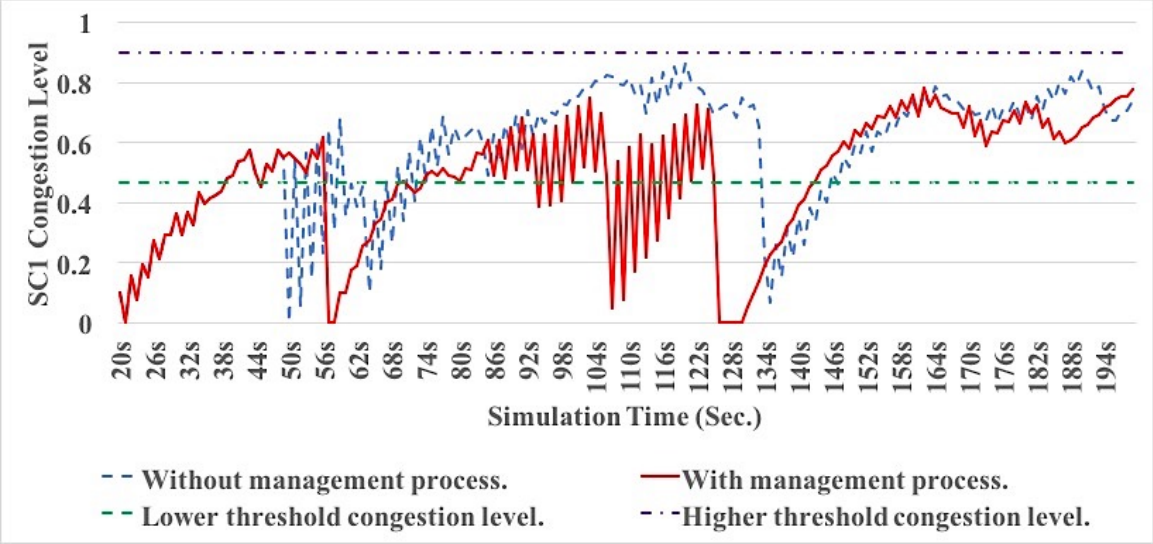


Figure 6-36, SC1 maximum congestion levels for both the DWFQ and the DRAM-NFV algorithms - case study 3, test scenario 3.

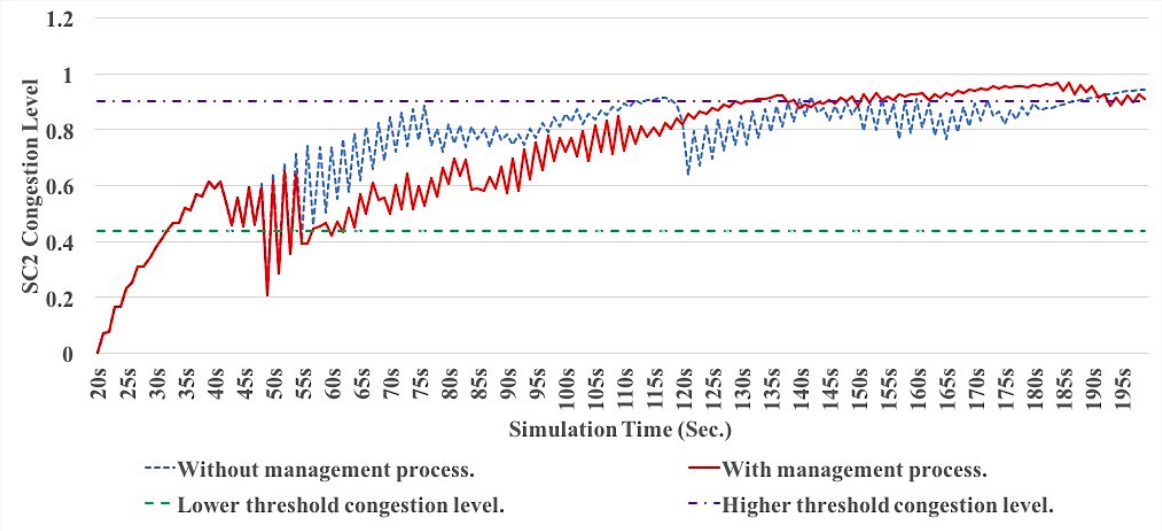


Figure 6-37, SC2 maximum congestion levels for both the DWFQ and the DRAM-NFV algorithms - case study 3, test scenario 3.

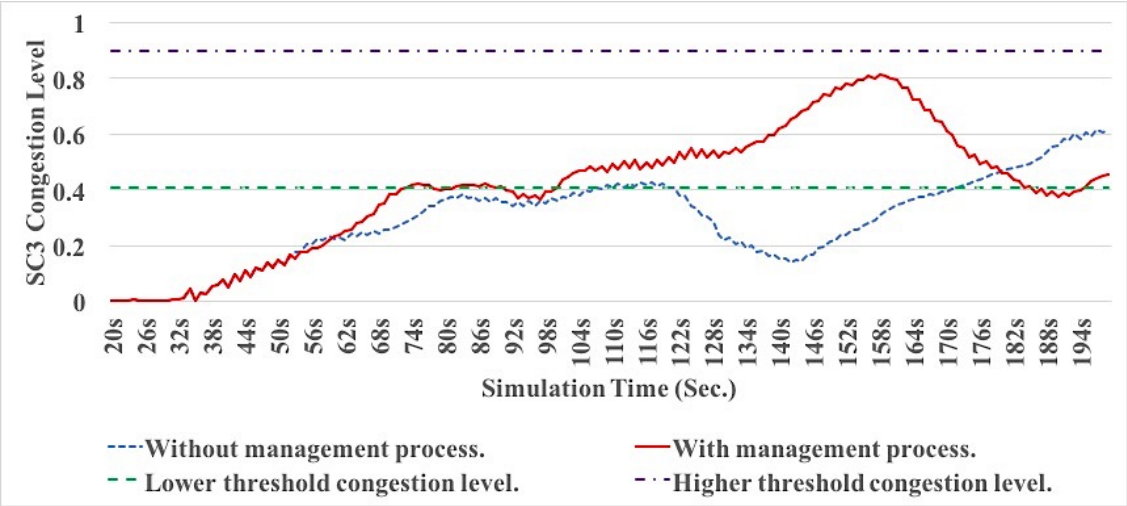


Figure 6-38, SC3 maximum congestion levels for both the DWFQ and the DRAM-NFV algorithms - case study 3, test scenario 3.

The DRAM-NFV algorithm organizes the utilization of the link that connects the upstream domain (Domain 1) with the downstream domain on the basis of the traffic conditions in the ingress routers of the downstream domain and the traffic conditions at the egress router of the upstream domain (Domain 1). It does improve some of the service class traffic utilization at that link when congestion occurs. The average percentage utilization for the highest and the lowest priority service classes (SC_1) and (SC_3) at that link are increased while the average percentage utilization for the medium priority service class (SC_2), which is considerably more congested than the other queues, is reduced. Table 6-30 illustrates the average percentage utilizations for the service classes traffic at the link that connects the upstream (Domain 1) and downstream domains for the whole period of simulation.

Table 6-30, Average percentage utilization for the service classes at the link that connects the upstream (Domain 1) and downstream DiffServ domains - Test scenario 3, case study 3.

Service Class	Domains Link utilization with DWFQ algorithm	Domains Link utilization with DRAM-NFV algorithm
SC1	6.563%	7.281%
SC2	45.141%	42.872%
SC3	0.4237%	0.618%

The upstream domain (Domain 3) in Figure 6-3 is not included in the process of resources management across domains as was said in the beginning of section 6.3.3. The egress router of (Domain 3) allocates its out-port resources for the service classes according to the service classes differentiation pattern and also according to the traffic conditions at its router without taking into consideration the traffic conditions at the ingress routers of the downstream domain. When comparing the service classes utilizations of the (Domain 1 – downstream) link with the service classes utilizations of the (Domain 3 – downstream) link (which uses the DWFQ algorithm only) the SC_1 and SC_2 traffic utilizations at the (Domain 1- downstream) link are better than the traffic utilizations for these service classes at the (Domain 3 – downstream) link although both upstream domains have the same configurations of service class queues and the same quantities of service classes traffic sources at (n_0) and (n_1) respectively. Table 6-31 summarizes the average percentage utilizations for the service classes traffic at the links that connect the upstream and downstream domains when managing the upstream (Domain 1) only using the DRAM-NFV algorithm for the whole period of simulation.

Table 6-31, Average percentage utilization for the service classes at the links that connect the upstream and downstream DiffServ domains - Test scenario 3, case study 3.

Upstream (Domain 1) – Downstream Link Utilization			Upstream (Domain 3) – Downstream Link Utilization	
Service Class	with DWFQ algorithm	with DRAM-NFV algorithm	with DWFQ algorithm	(Domain 3) – Downstream Link is managed using DWFQ algorithm while (Domain 1) – Downstream Link is managed using DRAM-NFV algorithm
SC1	6.563%	7.281%	7.861%	6.756%
SC2	45.141%	42.281%	39.229%	41.972%
SC3	0.423%	0.618%	0.431%	0.619%

The utilizations of some of the service class traffic at the destination link of the downstream domain are also improved with the use of the DRAM-NFV algorithm. The utilizations for the SC_1 and SC_3 traffic are increased while the utilization for the SC_2 traffic decreases as illustrated in Table 6-32 .

Table 6-32, Average percentage utilization for the service classes at the destination link when manging the upstream domain (Domain 1) only - Test scenario 3, case study 3.

Service Class	Destination Link utilization with DWFQ algorithm	Destination Link utilization with DRAM-NFV algorithm
SC1	2.02%	2.08%
SC2	20.955%	20.147%
SC3	0.407%	0.693%

In terms of service class average End to End Delays, the DRAM-NFV algorithm reduces the average End to End Delay for the SC_1 and SC_3 traffic and causes a little extra delay for the SC_2 traffic as shown in Table 6-33. The throughputs and average End to End Delays for the highest and lowest priority service classes (SC_1) and (SC_3) are improved when congestion occurs in the SC_2 queues at the ingress routers of the downstream domain by applying the DRAM-NFV algorithm.

Table 6-33, Average End-to-End Delay when manging the upstream domain (Domain 1) only, test scenario 3 - case study 3.

Service Class	Average End to End Delay using DWFQ algorithm	Average End to End Delay using DRAM-NFV algorithm	Increased/Decreased
SC1	6.112 sec	6.073 sec	Decreased by 0.04 sec.
SC2	6.895 sec	7.148 sec	Increased by 0.25 sec.
SC3	116.537 sec	94.987 sec	Decreased by 21.55 sec.
Overall traffic	8.734 sec	9.561 sec	Increased by 0.827 sec.

6.9 Chapter Summary.

This chapter presents the performance of the DRAM-NFV algorithm in managing resources across the edge routers of DiffServ domains. The DiffServ domain is represented by the simulated queue model application (RDWQueue) - section 5.3 , the traffic model application (TosApp) - section 6.2 and the network topology model – appendix A-5. The behaviour of the Proportional Delay DiffServ domain in term of classification the generated traffic into classes, service class queues configurations, average service class queue delays and service classes weights configurations is discussed in section 6.5. Moreover, the performance of the DRAM-NFV algorithm in managing resources across different DiffServ domains in the event of congestion is also tested and discussed in this chapter. This is done by applying this algorithm in the test scenarios which have been implemented using a simulation environment called "network simulator NS3" and measuring: i) the utilization of the link that connects the DiffServ domains, ii) the utilization of the link that connects the downstream domain to the destination and iii) measuring the average End to End Delay for service classes' traffic. The management is based on two factors. Firstly, reducing the scheduling rates of the congested service class queues at the egress router of the upstream domain based on the corresponding service class queues congestion levels at the ingress router of the downstream domain. Secondly, reallocating the resources of the link that connects the DiffServ domains based on the updated scheduling rates.

The algorithm is particularly effective when both the DiffServ domains (upstream and downstream) are underutilized but some of the service class queues at the ingress router of the downstream domain are either congested or suffering from various levels of congestion, while

other classes are not congested. Consequently, several case studies are presented for each test scenario; each case study includes increasing the traffic of (a) particular service class(es) at the upstream domain to create congestion states in the service class queues at the ingress routers of the downstream domain.

The results show that with dynamic management of resources across different DiffServ domains, the congestion levels in the service class queues at the ingress router of the downstream domain can be reduced. In addition, the algorithm prevents any congested service class queue at the egress router of the upstream domain from occupying the required bandwidth for its congested traffic and provides extra resources to other, uncongested, classes when such congestion occurs. By reallocating the resources of the link that connects DiffServ domains, the utilization of that link and the link utilization at the destination for some service class queues when congestion is present can be improved. Moreover, the average End to End Delay for some service class queues can also be improved. In contrast, the classes that suffered from congestion will reduce their traffic utilizations at these abovementioned links and increase their average End to End Delays.

Chapter Seven
Conclusions and Recommendations

7.1 Introduction:

In this research, a new resources allocation algorithm (DRAM-NFV) is proposed to allocate service class resources within proportional delay DiffServ domains. The DRAM-NFV algorithm manages the resources among service classes within the edge routers of the DiffServ domains dynamically according to their traffic conditions and manages these resources across different DiffServ domains, particularly in the event of congestion, based on the corresponding traffic conditions at the egress routers of the upstream domain and the ingress routers of the downstream domains. In this chapter, conclusions concerning this study along with a discussion of possible future research is presented.

7.2 Conclusions:

The importance of Quality of Service (QoS) technologies has increased rapidly with the rapid development of multimedia applications over the Internet such as distance learning, Video traffic, the voice over Internet protocol, IP TV and so on. There are different types of QoS models, and Differentiated Services (DiffServ) is one of them. It aims to achieve differentiation among service classes in the network through a sequence of operations such as classification, marking, queuing or dropping and scheduling based on the QoS requirements of the Internet applications. In addition, it has Per Hop Behaviour (PHB), which means that each domain or router manages its resources locally depending on its traffic condition and in a different way from other domains. The problem that we have in DiffServ is that it is static. This means that each domain allocates its resources separately according to domain configuration parameters. Once the domain administrator sets the configuration parameters of a domain, the parameters will not be changed. The traffic condition within a DiffServ domain can be changed continuously and randomly and therefore, there will be a situation where the network favours a certain amount of traffic while other network traffic will suffer despite the priority levels that we have made for service classes in a DiffServ domain so it needs to adjust the parameters at one domain to rebalance the resources at another domain. This research aims to achieve dynamic resource management for resources across and within DiffServ domains through using

the SDN and NFV technologies rather than the current static approach in allocating its resources separately per each domain in order to orchestrate resources between DiffServ domains.

The DiffServ is PHB and uses a static approach to allocating resources separately per each domain such that the domain administrator sets the parameters for service classes in a fixed manner and did not change it. However, it would be better if the DiffServ domains were managing resources dynamically rather than statically because the network operation (traffic condition) within a DiffServ domain could be changed continuously and therefore there would be a situation whereby the network favours a certain amount of network traffic. As a consequence, other network traffic will suffer. Instead of having a static DiffServ, we can monitor traffic condition in one domain and make changes to managing resources of other domains so we can now calibrate between DiffServ domains in managing resources. This calibration can be accomplished by programming remotely and automatically the OpenFlow switches of DiffServ domains, which are SDN devices, and by extracting one of the DiffServ network functions, which is resources management function, and implementing it virtually using the concept of Network Function Virtualisation NFV in order to deploy resources management across DiffServ domains dynamically when the traffic condition in a DiffServ domain is unbalanced; i.e.: when there is traffic congestion in a service class queue of a DiffServ domain. The management will still be local if the traffic condition in a DiffServ domain is balanced or normal.

The intended contribution of this research is to introduce a new scheduling algorithm called "Dynamic Resource Allocation Management - Network Function Virtualization (DRAM-NFV)" to manage the service class resources within and across the DiffServ domains in a dynamic and consolidated manner through using the SDN and NFV technologies. This new algorithm enables the Internet Service Providers (Administrators of DiffServ domains) to manage the resources of their Wide Area Networks (WANs) during the congestion periods in their networks.

The presented algorithm, which is run on the NFV server and uses SDN technology, takes into consideration all the possibilities of congestion occurring in a service class queue, and as we have SDN networks, we can monitor and manage the queues in a DiffServ domain remotely. The network statistics of a DiffServ domain like the average queue delay and average queue length need to be sent to the NFV server. After that, the NFV server calculates the congestion level in each service class queue at the ingress router of the downstream domain. If there is no congestion, then the DiffServ domain controller sets the service class weights and these weights are set based on the domain administrator configuration. If there is congestion in one of the

service classes, then NFV server calculates the scheduling rate update factor β and updates the scheduling rate of the equivalent service class at the egress router of the upstream domain. The condition that I decide if the service class queue is congested or not depends on the depth or size of the service class queue. The parameters that we are using in managing resources between DiffServ domains are the threshold values of service class queues, which are used to define the lower and higher threshold congestion levels. Average service class queue size is used to measure the congestion level α in a service class queue as a ratio of average queue size to service class buffer size and the scheduling rate update factor β. The job of the NFV server is to decide how much to reduce the service class scheduling rate between DiffServ domains. If the calculated congestion level is less than the low threshold value, then there is no need to reduce the scheduling rate of the equivalent service class at the egress router of the upstream domain; its rate is set based on the domain administrator configuration. If it is between the low and high threshold values then the scheduling rate is reduced by β. The relation between β and congestion level is linear such that the value of β is directly proportional to the value of the congestion level. If the service class is very congested, more so than the high threshold value, then β is set to the maximum value in order to dampen the service class throughput at the egress router of the upstream domain. A number of test scenarios are presented in this research in order to test the performance of DRAM-NFV algorithm. These scenarios simulate real world networks. The Network Simulator NS3 is used to simulate these scenarios. In order to evaluate the DRAM-NFV algorithm, the profile of VoIP, video traffic, FTP and Database request messages are considered when generating traffic in NS3. The algorithm is particularly effective when the DiffServ domains are underutilised but some of the service class queues at the ingress router of the downstream domain are either congested or suffering from different levels of congestion while other classes are not congested. The performance of the DRAM-NFV algorithm in managing resources across DiffServ domains is compared with the performance of the DWFQ algorithm, which manages resources within the edge routers of a DiffServ domain but cannot manage resources across DiffServ domains through looking at the utilization of the link that connects DiffServ domains, the utilization of the link that connects the downstream domain to the destination and the average End to End Delay for service classes traffic.

Managing resources dynamically across DiffServ domains instead of the current static approach to managing resources per each domain separately achieves better balance for DiffServ domains resources through monitoring the bandwidth hungry service class at one domain and managing its resources at another domain. As a consequence of this, the utilizations

of some service classes in the simulated test scenarios are improved. The second achievement when managing resources across DiffServ domains is that the average End to End Delay for overall traffic in the simulated test scenarios are reduced.

In test scenario 1- Case Study 1, the medium priority service class traffic SC2 is increased at the upstream domain. As a result, the SC1 and SC2 queues at ingress router of the downstream domain suffer from different levels of congestion. When managing resources dynamically using DRAM-NFV algorithm between the upstream and downstream domains, the average utilization for the SC2 and SC3 for the whole period of simulation at the destination end are increased by 0.23% and 0.035% respectively. The average End to End Delay for overall traffic is reduced by 700 msec.

In test scenario 2- Case Study 1, the lowest priority service class traffic SC3 is increased at the upstream domain. As a result, all queues at the ingress router of the downstream domain suffer from different levels of congestion. When managing resources dynamically using DRAM-NFV algorithm between the upstream and downstream domains, the average utilizations for the SC1 and SC2 for the whole period of simulation at the destination end are increased by 0.1% and 0.02% respectively. The average End to End Delay for overall traffic is also reduced by 700 msec.

While in test scenario 3- Case Study 2, the lowest priority service class traffic SC3 is increased at the upstream domain. As a result, the SC2 and SC3 queues at ingress router of the downstream domain are more congested than SC1 queue. When managing resources dynamically using DRAM-NFV algorithm between the upstream and downstream domains, the average utilization for the SC1 for the whole period of simulation at the destination end is increased by 0.175% and the average End to End Delay for overall traffic is also reduced by 800 msec.

As a result of reducing the average End to End Delay for overall traffic and improving the utilizations of service classes traffic, the QoS of applications traffic can be improved by dynamically managing the resources across DiffServ domains.

There are some limitations that were not taken into consideration when implementing a prototype of the DRAM-NFV algorithm using Network Simulator, NS3, and these will now be mentioned. This study did not consider the latency times involved in sending and receiving the management information to and from the Cloud infrastructure. Managing the resources across DiffServ domains can cause a few milliseconds delay because of sending the information related to traffic state from OpenFlow switches to NFV server for processing and sending back the configuration information to configure the queues of OpenFlow switches by the domain

controller. However, this delay can be trivial if the NFV server virtual resources, which are allocated for the SDN controllers, are enough for processing this information and configuring the queues quickly. The other limitation is that the marking strategies in all the simulated DiffServ domains are made to be identical, for simplicity. Considering different marking schemes within the DiffServ domains would increase the complexity of the Network Simulator NS3 programming and the analysis.

Throughout this thesis, we have made some recommendations for aspects of the research that would benefit from further attention and study. For instance, we could take the effect of a different parameter such as packet dropping in the equation of managing resources across DiffServ domains in order to improve the dynamic resource management process across DiffServ domains. The second recommendation is that the concept of this algorithm could be applied to other QoS models such as IntServ model or provide IntServ over DiffServ model. IntServ model needs to configure every router along a path and across domains from source to destination in order to reserve resources per each flow.

7.3 Review of Objectives:

During this research, several objectives have been achieved. These objectives are as follows:

1. The identification of the research problem relating to a common difficulty encountered by proportional delay DiffServ domains. Section 1.3 of this thesis described the research problem.
2. The reviewing of the algorithms that are used in allocating the resources within the edge routers of a proportional delay DiffServ domain, explaining the use of SDN to achieve better QoS, and the use of NFV. These topics were discussed in sections 3.2 and 3.3 respectively.
3. The presenting of a new algorithm for the dynamic and consolidated management of the resources within and across proportional delay DiffServ domains when traffic congestion occurs. This algorithm is called “Dynamic Resource Allocation Management – Network Function Virtualization (DRAM-NFV)”. The principle of the DRAM-NFV algorithm and the conditions which enable the DRAM-NFV algorithm to manage resources across DiffServ domains are identified in section 4.2 of this thesis.
4. The presenting of a mathematical model for the DRAM-NFV algorithm as illustrated in section 4.4 of this thesis.

5. The explaining of the simulated queueing model which is used to implement the proportional delay DiffServ and DRAM-NFV algorithm as illustrated in section 5.3 of this thesis.
6. The clarifying of the experimental design procedures which were used when evaluating and validating the DRAM-NFV algorithm. Section 6.3 of this thesis presents the test scenarios, and their case studies, which were used for this evaluation.
7. The discussion of the simulation results of the test scenarios presented, as exhibited in sections 6.6, 6.7 and 6.8.

7.4 Chapter Summary:

In this chapter, a comprehensive and precise overview of the research undertaken for this thesis has been presented. In addition, some specific issues have been listed in this chapter which deserves further research. This further research would complete this study and develop a cutting-edge infrastructure for the future Internet.

8. References

1. Al-Quzweeni, A. et al., 2016. A framework for energy efficient NFV in 5G networks. In *2016 18th International Conference on Transparent Optical Networks (ICTON)*. IEEE, pp. 1–4. Available at: http://ieeexplore.ieee.org/document/7550698/.
2. Alipio, M.I. et al., 2016. Demonstration of Quality of Service mechanism in an OpenFlow testbed. In *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. IEEE, pp. 443–447. Available at: http://ieeexplore.ieee.org/document/7867251/.
3. An, N. et al., 2016. Dynamic priority-adjustment for real-time flows in software-defined networks. In *2016 17th International Telecommunications Network Strategy and Planning Symposium (Networks)*. IEEE, pp. 144–149. Available at: http://ieeexplore.ieee.org/document/7751167/.
4. Armbrust, M. et al., 2009. *"Above the Clouds : A Berkeley View of Cloud Computing", technical report, University of California*,
5. Ashon, S.A. & Ilyas, M., 2011. *Cloud Computing and Software Services Theory and Techniques*,
6. Batalle, J. et al., 2013. On the Implementation of NFV over an OpenFlow Infrastructure: Routing Function Virtualization. In *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*. IEEE, pp. 1–6. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6702546 [Accessed June 5, 2014].
7. Bennett, J.C.R. & Zhang, H., 1997. Hierarchical packet fair queueing algorithms. *IEEE/ACM Transactions on Networking*, 5(5), pp.675–689.
8. Blake, S. et al., 1998. *An architecture for differentiated services*, Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:No+Title#0 [Accessed July 2, 2014].
9. Bodin, U. & Schelcn, O., 2001. Drop Strategies and Loss-rate Differentiation. , pp.146–154.
10. Buyya, R., Vecchiola, C. & Selvi, S.T., 2013. *Mastering Cloud Computing Foundations and Applications Programming*, Elsevier.
11. Carpenter, B. & Faucheur, F. Le, 2000. *"Per Hop Behavior Identification Codes", RFC 2836*,
12. Chen, N. et al., 2015. Self-Organizing Scheme Based on NFV and SDN Architecture for Future Heterogeneous Networks. *Mobile Networks and Applications*, 20(4), pp.466–472. Available at: http://link.springer.com/10.1007/s11036-015-0630-3.
13. Chin-Chang Li et al., 2000. Proportional delay differentiation service based on weighted fair queuing. In *Proceedings Ninth International Conference on Computer Communications and Networks (Cat.No.00EX440)*. IEEE, pp. 418–423. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=885523.
14. Chiosi, M. et al., 2012. *Network Functions Virtualisation - Introductory White Paper*,
15. Chowdhury, N.M.M.K. & Boutaba, R., 2009. Network virtualization: state of the art and research challenges. *IEEE Communications Magazine*, 47(7), pp.20–26. Available at: http://ieeexplore.ieee.org/document/5183468/.
16. Cisco, 2013. WRED and Queue Limit. Available at: http://www.cisco.com/c/en/us/td/docs/routers/10000/10008/configuration/guides/qos/qoscf/10qqueue.html#wp1041848.
17. Cisco Systems, I., 2006. *IPv6 Extension Headers Review and Considerations.*, Available at:

http://www.cisco.com/en/US/technologies/tk648/tk872/technologies_white_paper0900aecd8054d37d.html.
18. Clark, D.D. & Fang, W., 1998. Explicit allocation of best-effort packet delivery service. *IEEE/ACM Transactions on Networking*, 6(4), pp.362–373. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=720870 [Accessed June 25, 2014].
19. Demers, a., Keshav, S. & Shenker, S., 1989. Analysis and simulation of a fair queueing algorithm. *ACM SIGCOMM Computer Communication Review*, 19(4), pp.1–12.
20. Dong Xu, 2010. Cloud Computing: An emerging technology. In *2010 International Conference On Computer Design and Applications*. IEEE, pp. V1-100-V1-104. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5541105 [Accessed July 6, 2014].
21. Dovrolis, C. & Ramanathan, P., 1999. A case for relative differentiated services and the proportional differentiation model. *IEEE Network*, 13(5), pp.26–34. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=793688.
22. Dovrolis, C., Stiliadis, D. & Ramanathan, P., 2002. Proportional differentiated services: Delay differentiation and packet scheduling. *IEEE/ACM Transactions on Networking*, 10(1), pp.12–26. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=986503.
23. Dovrolis, C., Stiliadis, D. & Ramanathan, P., 1999. Proportional differentiated services: Delay Differentiation and Packet scheduling. *ACM SIGCOMM Computer Communication Review*, 29(4), pp.109–120. Available at: http://portal.acm.org/citation.cfm?doid=316194.316211.
24. Dovrolis, K., 2000. *Proportional differentiated services for the Internet*.
25. Duan, Q., Yan, Y. & Vasilakos, A. V., 2012. A survey on service-oriented network virtualization toward convergence of networking and cloud computing. *IEEE Transactions on Network and Service Management*.
26. ETSI, ETSI NFV Technology. *ETSI*. Available at: www.etsi.org/technologies-clusters/technologies/nfv.
27. Floodlight Project, 2013. Floodlight Project. Available at: http://www.projectfloodlight.org/floodlight/.
28. Flowgrammable Research Team, 2013. Message layer in OpenFlow. Available at: http://flowgrammable.org/sdn/openflow/message-layer/#tab_ofp_1_4.
29. Floyd, S. & Jacobson, V., 1995. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4).
30. Foster, I. et al., 2008. Cloud Computing and Grid Computing 360-degree compared. In *Grid Computing Environments Workshop, GCE 2008*.
31. Giordano, S. et al., 2003. Advances QoS provisioning in IP networks: The European premium IP projects. *IEEE Communications Magazine*, 41(1), pp.30–36.
32. Guck, J.W. & Kellerer, W., 2014. Achieving end-to-end real-time Quality of Service with Software Defined Networking. In *2014 IEEE 3rd International Conference on Cloud Networking, CloudNet 2014*.
33. Han, B. et al., 2015. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2), pp.90–97.
34. Han, Y. et al., 2016. An intent-based network virtualization platform for SDN. In *2016 12th International Conference on Network and Service Management (CNSM)*. IEEE, pp. 353–358. Available at: http://ieeexplore.ieee.org/document/7818446/.
35. Hawilo, H. et al., 2014. NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC). *IEEE Network*, 28(6), pp.18–26. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6963800.

36. Heinanen, J. et al., 1999. *"Assured Forwarding PHB Group", RFC 2597*,
37. Hill, R. et al., 2013. *Guide to cloud computing: principles and practice*, Springer.
38. Hu, C., Wang, Q. & Dai, X., 2015. SDN over IP: Enabling Internet to Provide Better QoS Guarantee. In *Proceedings - 2015 9th International Conference on Frontier of Computer Science and Technology, FCST 2015*.
39. IETF, 2013. *RFC7047*,
40. J. Babiarz, K. Chan, F. Baker, Nortel Networks, C.S., 2006. *RFC 4594 - Configuration Guidelines for DiffServ Service Classes*,
41. J. Virtamo, 2005. Queuing theory - priority queues. In *Queuing Theory*. Available at: https://www.netlab.tkk.fi/opetus/s383143/kalvot/E_priority.pdf.
42. Jamjoom, H., Williams, D. & Sharma, U., 2014. Don't call them middleboxes, call them middlepipes. In *Proceedings of the third workshop on Hot topics in software defined networking - HotSDN '14*. New York, New York, USA: ACM Press, pp. 19–24. Available at: http://dl.acm.org/citation.cfm?doid=2620728.2620760.
43. Karakus, M. & Durresi, A., 2017. Quality of Service (QoS) in Software Defined Networking (SDN): A survey. Available at: www.elsevier.com/locate/jnca.
44. Karaman, M.A. et al., 2015. Quality of service control and resource prioritization with Software Defined Networking. In *1st IEEE Conference on Network Softwarization: Software-Defined Infrastructures for Networks, Clouds, IoT and Services, NETSOFT 2015*.
45. Kate Greene, 2009. MIT Tech Review 10 Breakthrough Technologies: Software-defined Networking. Available at: http://www2.technologyreview.com/.
46. Kim, H. & Feamster, N., 2013. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2), pp.114–119. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6461195.
47. Kreutz, D. et al., 2015. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, 103(1), pp.14–76. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6994333.
48. Kun I. Park, 2005. *QoS in Packet Networks*, Springer.
49. Kurose, J.F. & Ross, K.W., 2013. *COMPUTER NETWORKING A Top-Down Approach* Sixth., PEARSON.
50. Kusmierek, E. et al., 2002. An integrated network resource and QoS management framework. In *IEEE Workshop on IP Operations and Management.* IEEE, pp. 68–72. Available at: http://ieeexplore.ieee.org/document/1045758/.
51. Language, C. programming, 2016. C++ online learning programming language. Available at: http://www.cplusplus.com.
52. Lee, C., Shin, S. & Chung, J.-M., 2016. Enhanced LTE handover scheme using NFV for LTE handover delay reduction. In *2016 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*.
53. Lin, P. et al., 2015. A west-east bridge based SDN inter-domain testbed. *IEEE Communications Magazine*, 53(2), pp.190–197.
54. Luo, S. et al., 2016. Improving Energy Efficiency in Industrial Wireless Sensor Networks Using SDN and NFV. In *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*. IEEE, pp. 1–5. Available at: http://ieeexplore.ieee.org/document/7504281/.
55. M.P.V. Manthena, 2015. *Network-as-a-Service Architecture with SDN and NFV*.
56. McKeown, N. et al., 2008. OpenFlow. *ACM SIGCOMM Computer Communication Review*, 38(2), p.69. Available at: http://doi.acm.org/10.1145/1355734.1355746%5Cnhttp://portal.acm.org/citation.cfm?id=1355746.
57. Mell, P. & Grance, T., 2011. The NIST Definition of Cloud Computing

Recommendations of the National Institute of Standards and Technology. *Nist Special Publication*, 145, p.7. Available at: http://www.mendeley.com/research/the-nist-definition-about-cloud-computing/.
58. Moret, Yan, and S.F., 1998. A proportional queue control mechanism to provide differentiated services. In *Ptoceedings of the Internation Symposium on Computer and Information Systems (ISCIS)*.
59. Nichols, H. et al., 1998. *"Definition of the Differentiated Services Filed (DiffServ Field) in the IPv4 and IPv6 Headers", RFC 2474*,
60. ns-2 lists contributors., 1995. The Network Simulator - ns-2. Available at: http://www.isi.edu/nsnam/ns/.
61. NS3 Doxygen, 2015. ns-3 Direct Code Execution Documentation (Doxygen). Available at: https://www.nsnam.org/docs/release/3.23/doxygen/index.html.
62. NS3 Manual, 2015. ns-3 Manual.
63. NS3 Model library, 2015. ns-3 Model Library.
64. ns3 project, 2010. Procedure of adding new module to ns3 packetge. Available at: https://www.nsnam.org/docs/manual/html/new-modules.html.
65. NS3 software users group., NS3 Users Forum. Available at: https://groups.google.com/forum/#!forum/ns-3-users.
66. NS3 Tutorial, 2015. ns-3 Tutorial.
67. ONOS Project, 2015. Open Source Network Operating System. Available at: http://onosproject.org/.
68. Open Networking Foundation (ONF), 2014a. *of-config-1.2 OpenFlow Management and Configuration Protocol*, Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf.
69. Open Networking Foundation (ONF), 2011. *OpenFlow Switch Specification version 1.2*,
70. Open Networking Foundation (ONF), 2012. *OpenFlow Switch Specification version 1.3*,
71. Open Networking Foundation (ONF), 2013. *OpenFlow Switch Specification version 1.4*,
72. Open Networking Foundation (ONF), 2014b. *SDN architecture*, Available at: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf.
73. Open Networking Foundation (ONF), 2014. *OpenFlow Switch Specification Version 1.5*,
74. OpenDaylight Project, 2013. OpenDaylight Project. *OpenDaylight Project*. Available at: https://www.opendaylight.org.
75. Palma, D. et al., 2014. The QueuePusher: Enabling queue management in OpenFlow. In *Proceedings - 2014 3rd European Workshop on Software-Defined Networks, EWSDN 2014*.
76. Panza, G., Grazioli, M. & Sidoti, F., 2006. Design and analysis of a dynamic Weighted Fair Queuing (WFQ) scheduler. *IST M obile and W ireless Communication Summit '05*.
77. Pfaff Ben, Bob Lantz, B.H., 2011. *OpenFlow Switch Specification version 1.1*, Available at: http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf.
78. Pfaff Ben, Brandon Heller, D.T., 2009. *OpenFlow Switch Specification version 1.0*, Available at: http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf.
79. Raghavan, B. et al., 2012. Software-defined internet architecture. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks - HotNets-XI*. New York, New York, USA: ACM Press, pp. 43–48. Available at: http://dl.acm.org/citation.cfm?id=2390239.
80. Rao, S.K.N., 2014. *SDN AND ITS USE - CASES - NV AND NFV A State-of-the-Art Survey*,
81. Reichmeyer, F. & Networks, N., 1998. *A Two-Tier Resource Management Model for Differentiated Services Networks 1 Introduction : a High-Level Model of QoS Control*,

82. Shaikh, F.A. et al., 2002. End-to-end testing of IP QoS mechanisms. *Computer*, 35(5), pp.80–87. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=999779.
83. Sharma, R., Kumar, N. & Talabattula, S., 2014. Performance of new dynamic benefit-weighted scheduling scheme in DiffServ networks. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, pp. 2578–2583. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6968380.
84. Soares, J. et al., 2014. Cloud4NFV: A platform for Virtual Network Functions. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*. IEEE, pp. 288–293. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6969010.
85. Srinivasan, V., Suri, S. & Varghese, G., 1999. Packet classification using tuple space search. *ACM SIGCOMM Computer Communication Review*, 29(4), pp.135–146. Available at: http://portal.acm.org/citation.cfm?doid=316194.316216.
86. Suter, B. et al., 1999. Design considerations for supporting TCP with per-flow queueing. In *Proceedings. IEEE INFOCOM '98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No.98CH36169)*. IEEE, pp. 299–306. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=659666.
87. Tanenbaum, A.S. & Wetherall, D.J., 2011. *Computer Networks* Fifth., Pearson. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:No+Title#0.
88. Theophilus Benson, A.A. and D.M., 2009. *Unraveling the Complexity of Network Management*, Berkeley,CA,USA. Available at: https://www.usenix.org/legacy/events/nsdi09/tech/full_papers/benson/benson_html/.
89. Tim Szigeti, and C.H., 2005. *End-to-End QoS Network Design: Quality of Service in LANs, WANs, and VPNs.*, Cisco Press.
90. Urgun, Y. & Kavak, A., 2016. SDN/NFV-based QoS mechanism for Cellular Core Networks. In *2016 24th Telecommunications Forum (TELFOR)*. IEEE, pp. 1–4. Available at: http://ieeexplore.ieee.org/document/7818733/.
91. Vaughan-Nichols, S.J., 2011. TECHNOLOGY NE WS OpenFlow: The Next Generation of the Network?
92. Volvach, I. & Globa, L., 2016. Mobile networks disaster recovery using SDN-NFV. In *2016 International Conference Radio Electronics & Info Communications (UkrMiCo)*. IEEE, pp. 1–3. Available at: http://ieeexplore.ieee.org/document/7739648/.
93. Wallner, R. & Cannistra, R., 2013. An SDN Approach: Quality of Service using Big Switch's Floodlight Open-source Controller. *Proceedings of the Asia-Pacific Advanced Network*, 35, pp.14–19. Available at: http://dx.doi.org/10.7125/APAN.35.2.
94. Wang, H.W.H., Shen, C.S.C. & Shin, K.G., 2001. Adaptive-weighted packet scheduling for premium service. *ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No.01CH37240)*, 6, pp.1846–1850.
95. WANG, Z., 2001. *Internet QoS Architectures and Mechanisms for Quality of Services.*, Elsevier.
96. Wendell Odom, and M.J.C., 2005. *Cisco QoS Exam Certification Guide*, Cisco Press.
97. Wibowo, F.X.A. et al., 2017. Multi-domain Software Defined Networking: Research status and challenges. Available at: www.elsevier.com/locate/jnca.

9. Appendixes

A-1 Research Problem Experiment:

In order to present the research problem, an experiment has been carried out using NS3 simulator using the scenario shown in Figure 9-1. In this experiment, we call the upstream (in terms of direction of traffic) domain, Domain 1, and the downstream domain, Domain 2. Both are relative DiffServ domains. The marking strategy for both domains is assumed to be identical for simplicity; both provide three types of service classes according to their priorities from the highest to lowest priorities: SC_1, SC_2 and SC_3 such that SC_1 is dedicated for the VoIP traffic, SC_2 is dedicated for video traffic and SC_3 is dedicated for FTP traffic. The core router of the downstream domain connects with multiple LANs and these networks are injected with best effort traffic in order to modify the network traffic conditions of the downstream domain. In this simulation, best effort traffic can be inserted into any defined service class queue at the downstream domain, depending on the available buffer resources at these queues. All the queues which belong to a specific service class in both the upstream and downstream DiffServ domains in this experiment are assumed to have the same queue configuration in terms of queue buffer size, threshold parameters and drop probabilities for the packet drop algorithm (WRED) as shown in Table 9-1. The buffer sizes of these abovementioned service class queues can be expressed in terms of power of 2 numbers of packets: i.e., (64, 128, 256, 512) packets (Wendell Odom 2005) so the buffer sizes of these classes are configured to 128 packets in both domains and the WRED configuration of the Cisco™ [6] 10000 series QoS routers (Cisco 2013) is used in this experiment to configure the threshold values and drop probabilities for these service classes. Table 9-2 illustrates the configuration parameters of the research problem experiment.

Table 9-1, Explanation of research problem - service classes configuration parameters.

	Buffer Size (in term of number of packets)	Minimum Threshold (times the queue size)	Maximum Threshold (times the queue size)	Drop Probability
Highest Priority (SC_1)	128	15/32	½	1/10
Medium Priority (SC_2)	128	7/16	½	1/10
Lowest Priority (SC_3)	128	13/32	½	1/10

[6] Cisco™ is a trade mark of Cisco Systems, Inc. and its affiliates.

Table 9-2, Configuration parameters of the research problem experiment.

Parameter	Configuration value	Notes
Link capacities of the upstream domain	2 Mbits	To ensure that more DiffServ traffic is injected into the downstream domain from the upstream domain.
Link capacity that connects the upstream and downstream domains.	4 Mbits	
link capacities of the downstream domain	2 Mbits	
Link capacity that connects the downstream domain and destination.	4 Mbits	
Both the upstream and downstream domains provide three service class queues	(SC_1, SC_2, SC_3)	
The delay configuration parameters (δ) for service classes (SC_1:SC_2:SC_3) at the edge DiffServ routers.	(1:2:4)	(Chin-Chang Li et al. 2000).
The service class markings in the upstream and downstream domains.	Identical.	
Service class queue buffer size in the upstream and downstream domains.	128 packets	(Wendell Odom 2005).
Minimum (lower) Service class queue threshold values (SC_1:SC_2:SC_3) in the upstream and downstream domains.	(15/32:7/16:13/32) times the queue size.	(Cisco 2013)
Service class queue dropping probabilities (SC_1:SC_2:SC_3) in the upstream and downstream domains.	(1/2:1/2:1/2) times the queue size.	(Cisco 2013)
Traffic Distribution in the upstream and downstream domains.	VoIP (SC1) – Highest Priority, Video Traffic (SC2), FTP (SC3) – Lowest Priority, and Data Base Request message (Best Effort).	

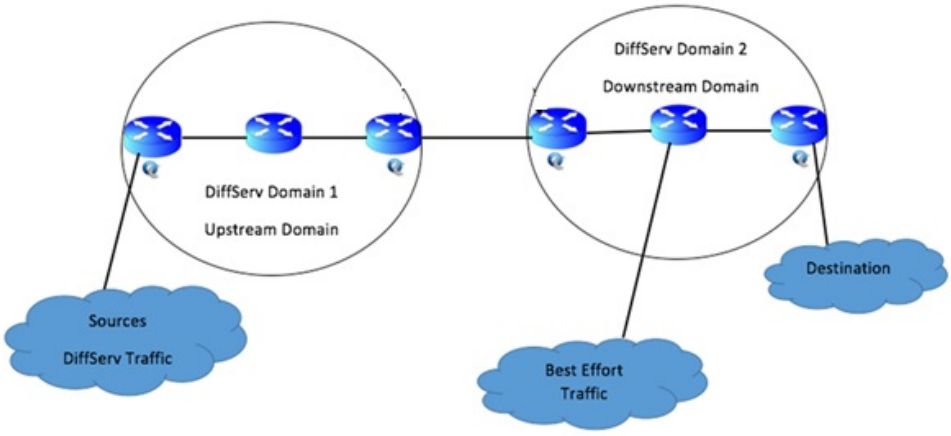


Figure 9-1, Research problem simulation scenario.

In this experiment, we increase the traffic of one of the service classes traffic at the upstream domain, the traffic of SC_2 (Video Traffic) for example. When the SC_2 traffic at the upstream domain of Figure 9-1 increases, the resources of the link that connects the upstream and downstream domains will (currently) be configured based only on the traffic conditions of the egress upstream router and may become mostly occupied by the increasing SC_2 traffic – plus the SC_1 and SC_3 traffic as shown in Figure 9-2, Figure 9-3 and Figure 9-4. In addition, the resources of the downstream domain may also become mostly occupied by the increasing SC_2 traffic (plus the SC_1 and SC_3 traffic). The average utilization of SC_2 traffic at the link that connects both domains is about 60% of the link capacity while the average utilization for the highest priority service class SC_1 is about 18% of the link capacity and for the lowest priority service class SC_3 is about 2.5% of the link capacity.

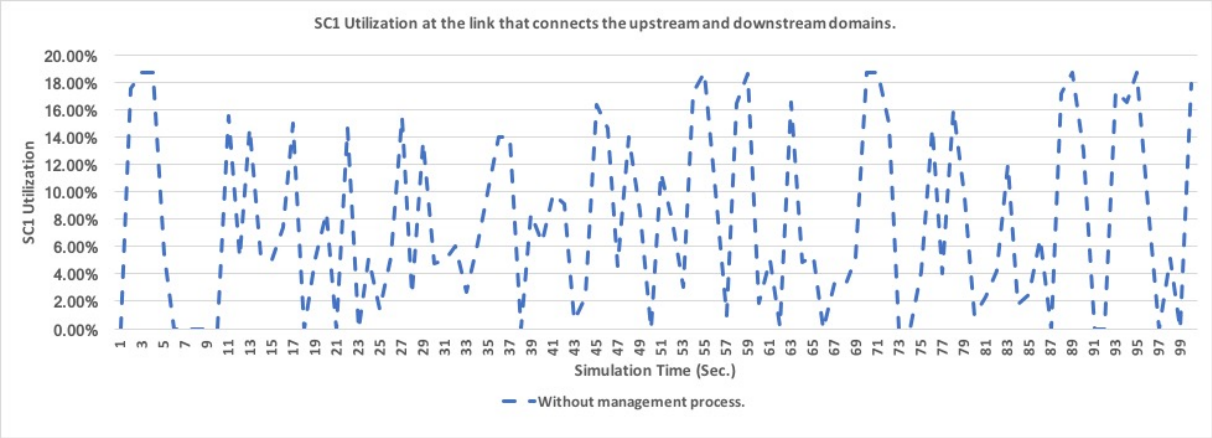


Figure 9-2, Explanation of research problem -SC1 utilization at the link that connects the upstream and downstream DiffServ domains.

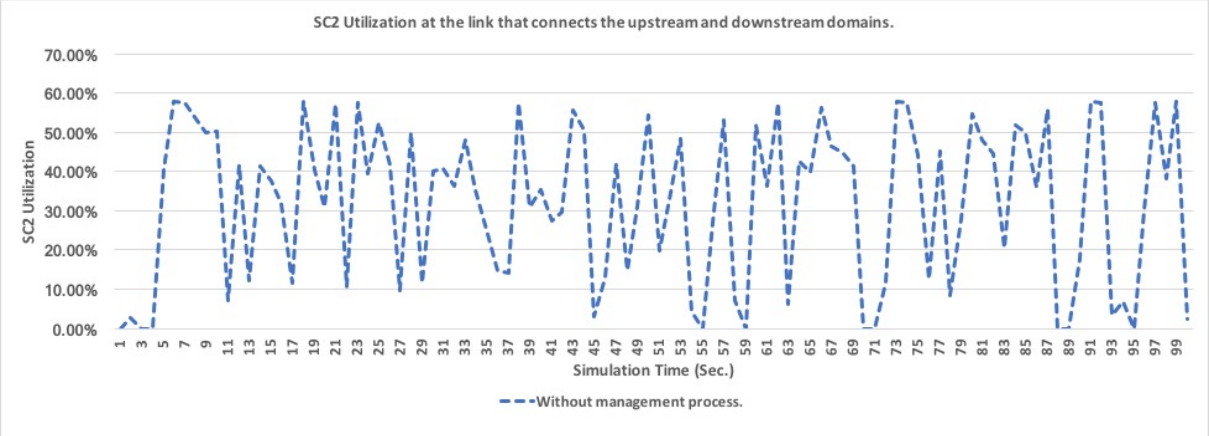


Figure 9-3, Explanation of research problem -SC2 utilization at the link that connects the upstream and downstream DiffServ domains.

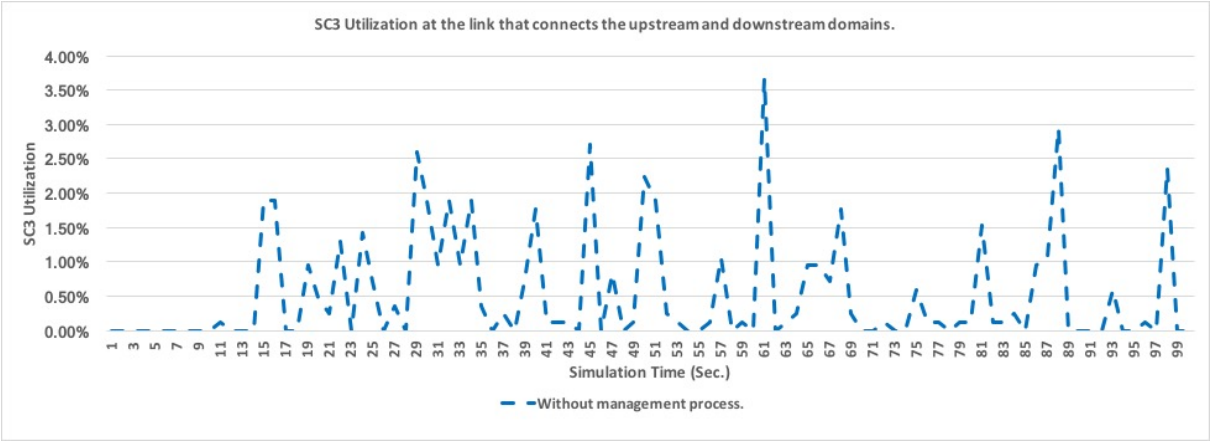


Figure 9-4, Explanation of research problem- SC3 utilization at the link that connects the upstream and downstream DiffServ domains.

In addition, there are still available resources not utilized well for SC_1 and SC_3 queues at the ingress router of the downstream domain while SC_2 queue resource is completely utilized. This means that the downstream domain favoring the medium priority traffic SC_2 while the traffic of SC_1 and SC_3 are suffering. Figure 9-5, Figure 9-6 and Figure 9-7 represent the congestion level in each service class queue at the ingress router of the downstream domain. The congestion level can be expressed as the ratio of the average service class queue size or length to the allocated service class buffer size. It is noticeable that SC_2 queue is congested at the ingress router of the downstream domain, its congestion level is about 0.8 of its buffer size for the whole period of simulation while the congestion level for the highest priority service class is between 0.4 and 0.7 of SC_1 buffer size for the whole period of simulation and for the lowest priority service class SC_3 do not exceed 0.3 of its buffer size for the whole period of simulation.

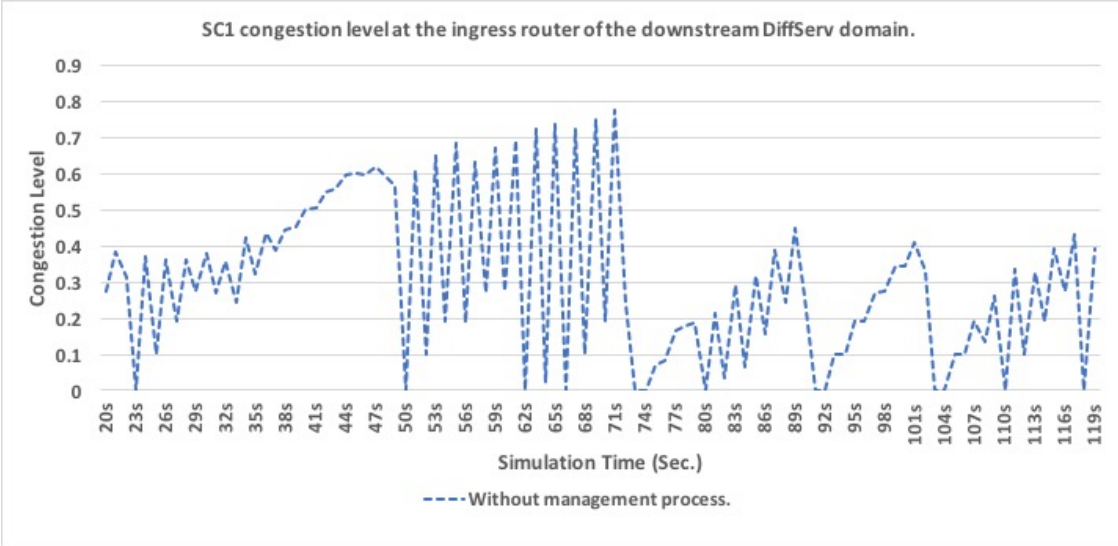


Figure 9-5, Explanation of research problem -SC1 congestion level at the ingress router of the downstream domain.

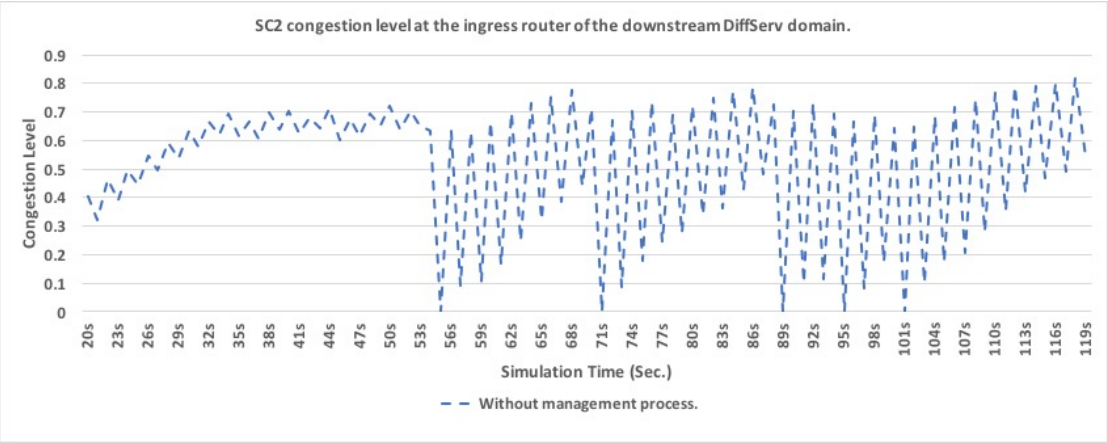


Figure 9-6, Explanation of research problem- SC2 congestion level at the ingress router of the downstream domain.

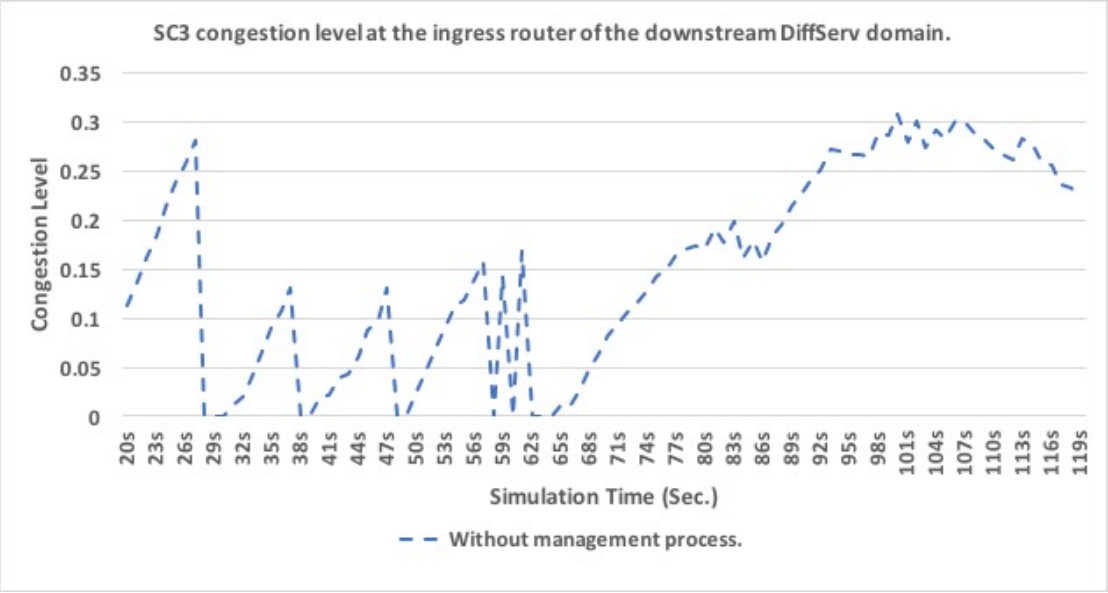


Figure 9-7, Explanation of research problem -SC3 congestion level at the ingress router of the downstream domain.

A-2: Use of NS3 Class References in the Simulated Queue and Traffic Model:

A-2.1: In RDWQueue Model:

Table 9-3, NS3 class references which are used to build the RDWQueue model and their purposes.

NS3 class reference	Purpose
NS3::Queue class reference	To build the RDWQueue model of a DiffServ.
NS3::Object model	To instantiate, identify and set the attributes of the instantiated RDWQueue model.
NS3::Simulator Class Reference	To schedule the events of RDWQueue.
NS3::Exponential Random Distribution Class Reference	To generate random value from exponential random distribution.
NS3::QoSTag Class Reference	To provide a means whereby applications which generate traffic can specify the application tag and define traffic flow

A-2.2: In TosApp Model:

Table 9-4, NS3 class references which are used to build the TosApp model and their purposes.

NS3 class reference	Purpose
NS3::Socket Class Reference	To create one or more TCP or UDP socket(s) for the source addresses that are to be connected to destinations for packet sending.
NS3::Address Class Reference	To specify the addresses of destinations.
NS3::Packet Class Reference	To create packets with the required packet size in bytes, add a packet tag (ToS) to the created packets and send the packets to these destinations.
NS3::QosTag Class Reference	To set-up the ToS tag for the packet to be sent.
NS3::Exponential Random Variable Class Reference	To obtain random inter-arrival times which are used in scheduling the events of sending packets and to obtain random but bounded packet sizes
NS3::Constant Random Variable Class Reference	To obtain constant inter-arrival times for scheduling the events of sending packets.
NS3::Data Rate Class Reference	To construct a data rate for the generated traffic in bits/sec which is used in scheduling the events of sending packets.
NS3::Simulator Class Reference	To schedule the events of sending the generated packets of TosApp.

A-3: Queue Model NS3 Simulation Code:

```
#ifndef RDWQUEUE_H
#define RDWQUEUE_H

#include <queue>
#include <deque>
#include <map>
#include <list>
#include <cmath>


#include "ns3/packet.h"
#include "ns3/queue.h"
#include "ns3/nstime.h"
#include "ns3/boolean.h"
#include "ns3/data-rate.h"
#include "ns3/nstime.h"
#include "ns3/ipv4-header.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/csma-module.h"

using namespace ns3;

class RDWQueue : public Queue
{
public:
    static TypeId GetTypeId(void);
    RDWQueue();
    RDWQueue(int id);
    virtual ~RDWQueue();
    // common status
    uint32_t linkID;          //!< which link this queue was installed
    DataRate m_linkBandwidth; //!< Link bandwidth
    Time m_linkDelay;         //!< Link delay
    uint32_t BYTEmode;
    uint32_t SC1Packets;
    uint32_t SC2Packets;
    uint32_t SC3Packets;
    uint32_t SC1DepartPackets;
    uint32_t SC2DepartPackets;
    uint32_t SC3DepartPackets;
    uint32_t SC4PKTsSC1;
    uint32_t SC4PKTsSC2;
    uint32_t SC4PKTsSC3;
    double_t usc1;  // edit 29/09
    double_t usc2;  // edit 29/09
    double_t usc3;  // edit 29/09
    double_t u;  // edit 29/09
    double_t UtiliSC1;
    double_t UtiliSC2;
    double_t UtiliSC3;

    uint32_t band;
    uint32_t Pk_ID;

    int currSecond;
    int NumberOfPktsPerSec;
    int NumberOfPktsPerK;
```

```
62
63      // customized status
64      void setPKTBUFFER(uint32_t sc1, uint32_t sc2, uint32_t sc3);
65      uint32_t SC1PKTBUFFER;
66      uint32_t SC2PKTBUFFER;
67      uint32_t SC3PKTBUFFER;
68      void setBYTEBUFFER(uint32_t sc1, uint32_t sc2, uint32_t sc3);
69      uint32_t SC1BYTEBUFFER;
70      uint32_t SC2BYTEBUFFER;
71      uint32_t SC3BYTEBUFFER;
72      void setREDTHIGH(double sc1, double sc2, double sc3);
73      double REDSC1THIGH;
74      double REDSC2THIGH;
75      double REDSC3THIGH;
76      void setREDTHIGHPKT(double sc1, double sc2, double sc3);
77      double REDSC1THIGHPKT;
78      double REDSC2THIGHPKT;
79      double REDSC3THIGHPKT;
80
81      void setREDHDROP(double sc1, double sc2, double sc3);
82      double REDSC1HDROP;
83      double REDSC2HDROP;
84      double REDSC3HDROP;
85      void setREDTLOW(double sc1, double sc2, double sc3);
86      void setREDTLOWPKT(double sc1, double sc2, double sc3);
87
88      void setALPHALPKT(float sc1, float sc2, float sc3);
89      void setALPHAHPKT(float sc1, float sc2, float sc3);
90
91      double REDSC1TLOW;
92      double REDSC2TLOW;
93      double REDSC3TLOW;
94      double REDSC1TLOWPKT;
95      double REDSC2TLOWPKT;
96      double REDSC3TLOWPKT;
97
98      void setREDLDROP(double sc1, double sc2, double sc3);
99      double REDSC1LDROP;
100     double REDSC2LDROP;
101     double REDSC3LDROP;
102     void setALPHAL(float sc1, float sc2, float sc3);
103     float SC1CONGESTIONFATORALPHAL;
104     float SC2CONGESTIONFATORALPHAL;
105     float SC3CONGESTIONFATORALPHAL;
106     float PKTSC1CONGESTIONFATORALPHAL;
107     float PKTSC2CONGESTIONFATORALPHAL;
108     float PKTSC3CONGESTIONFATORALPHAL;
109
110     void setALPHAH(float sc1, float sc2, float sc3);
111     float SC1CONGESTIONFATORALPHAH;
112     float SC2CONGESTIONFATORALPHAH;
113     float SC3CONGESTIONFATORALPHAH;
114     float PKTSC1CONGESTIONFATORALPHAH;
115     float PKTSC2CONGESTIONFATORALPHAH;
116     float PKTSC3CONGESTIONFATORALPHAH;
117
118     void setBELTA(float sc1, float sc2, float sc3);
119     float CONGESTIONFATBELTA1;
120     float CONGESTIONFATBELTA2;
121     float CONGESTIONFATBELTA3;
122
```

```
123       // queue mode is packet, for calculate the delay.
124
125       virtual bool DoEnqueue(Ptr<Packet> p);
126       virtual Ptr<Packet> DoDequeue(void);
127       int PktDeqScPromo;
128       virtual Ptr<const Packet> DoPeek(void) const;
129
130       uint32_t Classify(Ptr<const Packet> p); // check the tos
131       double GenRand(void);
132
133       float MeasureQueueLength(void); // measure the queue length every time
          interval, store in usagelist
134
135       std::list<double> SC1Usage;
136       std::list<double> SC2Usage;
137       std::list<double> SC3Usage;
138
139       // For average queue length calculation new method
140
141       std::vector<double> SC1PrevUsage;
142       std::vector<double> SC2PrevUsage;
143       std::vector<double> SC3PrevUsage;
144
145       std::vector<double> SC1CurreUsage;
146       std::vector<double> SC2CurreUsage;
147       std::vector<double> SC3CurreUsage;
148
149
150       std::vector<double> SC1CurreDelay;
151       std::vector<double> SC2CurreDelay;
152       std::vector<double> SC3CurreDelay;
153
154       double SC1CurrUsage;
155       double SC2CurrUsage;
156       double SC3CurrUsage;
157
158       float MeasureDelay(void); // measure the queue length every time interval,
          store in usageset
159
160       std::list<uint64_t> SC1Delay;
161       std::list<uint64_t> SC2Delay;
162       std::list<uint64_t> SC3Delay;
163
164       void CalculateSchedulingRate(void); // calculate everything in here.
165       int CalRatePktType;
166       float* CalculateSchedulingRateFixed(float rates[3]);
167       float* CalculateSchedulingRatePktType1(float rates[3]);
168       float* CalculateSchedulingRatePktType2(float rates[3]);
169       float* CalculateSchedulingRateByte(float rates[3]);
170
171       float SC1ScheRate;
172       float SC2ScheRate;
173       float SC3ScheRate;
174       bool CalculateBucket(float sc1, float sc2, float sc3);
175       int DeQMode1CurrCls;
176
177       double SC1ScheRateAvg;
178       double SC2ScheRateAvg;
179       double SC3ScheRateAvg;
180
181       uint32_t SC1BUCKETS;
```

```
182       uint32_t SC2BUCKETS;
183       uint32_t SC3BUCKETS;
184
185       uint32_t SC1BUCKETSCPY;
186       uint32_t SC2BUCKETSCPY;
187       uint32_t SC3BUCKETSCPY;
188       double Ran;
189
190       float GetSchedulingRate(int band);
191       bool SetSchedulingRate(int band, float SchRate);                // for node 4,8
192       bool SetSchedulingRateByFactor(int band, float alpha);          // for node 5
193       bool SetSchedulingRateByFactorPacketMode(int band, float alpha); // for node 5
194       bool SetSchedulingRateByFactorSC1;
195       bool SetSchedulingRateByFactorSC2;
196       bool SetSchedulingRateByFactorSC3;
197
198       bool CalculateWeight(void); // calculate the weight
199       float SC1Weight;
200       float SC2Weight;
201       float SC3Weight;
202       float GetWeight(int band);
203       bool SetWeight(int band, float Weight); // for node 4,8
204
205       std::deque<Ptr<Packet> > m_packets1; //!< the packets in band 0
206       std::deque<Ptr<Packet> > m_packets2; //!< the packets in band 1
207       std::deque<Ptr<Packet> > m_packets3; //!< the packets in band 2
208
209       std::map<uint64_t, Time> SC1EnqueuTime;
210       std::map<uint64_t, Time> SC2EnqueuTime;
211       std::map<uint64_t, Time> SC3EnqueuTime;
212       std::map<uint64_t, Time> SC1DequeuTime;
213       std::map<uint64_t, Time> SC2DequeuTime;
214       std::map<uint64_t, Time> SC3DequeuTime;
215
216       uint64_t sc1ByteLength; //!< Queue length in bytes(usage)
217       uint64_t sc2ByteLength;
218       uint64_t sc3ByteLength;
219
220       uint64_t sc1qavg; //!< Average Queue length in bytes(usage)
221       uint64_t sc2qavg;
222       uint64_t sc3qavg;
223
224      double REDSC1LD;
225      double REDSC2LD;
226      double REDSC3LD;
227
228       uint32_t sc1dropcount;
229       uint32_t sc2dropcount;
230       uint32_t sc3dropcount;
231       uint32_t sc4dropcount;
232
233       uint32_t sc1Hdropcount;
234       uint32_t sc1Ldropcount;
235       uint32_t sc1Bdropcount;
236
237       uint32_t sc2Hdropcount;
238       uint32_t sc2Ldropcount;
239       uint32_t sc2Bdropcount;
240
241       uint32_t sc3Hdropcount;
242       uint32_t sc3Ldropcount;
```

```
243       uint32_t sc3Bdropcount;
244
245       /**
246       * \brief Get the current value of the queue in bytes or packets.
247       *
248       * \returns The queue size in bytes or packets.
249       */
250       uint32_t GetQueueSizeBytes(uint32_t band);
251       uint32_t GetQueueSizePkts(uint32_t band);
252       void ShowStatisticReport(void);
253
254       double getAverageQueueLength(int band);
255       float getAverageQueueDelay(int band);
256       void CalculateSchedulingRateQ6(int band, double qlength, float delay);
257
258       uint32_t sc1pktsinsecond;
259       uint32_t sc2pktsinsecond;
260       uint32_t sc3pktsinsecond;
261
262       void UpdateLinkUtlization(uint32_t band, uint32_t pktsize);
263       void getLinkUtlization();
264
265       void UpdatePacketsPerSec(uint32_t band);
266       void getPktsPerSec();
267       uint32_t sc1pktsinsec;
268       uint32_t sc2pktsinsec;
269       uint32_t sc3pktsinsec;
270
271
272       //bool SetSchedulingRateByAverage(int band, float sc1averagelength, float       ↵
          averagelength, float averagedelay);
273       bool SetSchedulingRateByAverage(int band, double sc1averagelength, double      ↵
          sc2averagelength, double sc3averagelength, double sc1averagedelay,double      ↵
          sc2averagedelay, double sc3averagedelay );
274
275       bool SetSchedulingRateByLinearFactor(int band, float avgqueuelength);
276       bool SetSchedulingRateByLinearFactorSC1;
277       bool SetSchedulingRateByLinearFactorSC2;
278       bool SetSchedulingRateByLinearFactorSC3;
279
280       bool SetSchedulingRateByAverageSC1;
281       bool SetSchedulingRateByAverageSC2;
282       bool SetSchedulingRateByAverageSC3;
283
284       void UpdateAverageQueueDelay(int band, double AverageQueueDelay);
285       double GetAverageQueueDelay(int band);
286       std::vector<double> SC1CurrAverageDelay;
287       std::vector<double> SC2CurrAverageDelay;
288       std::vector<double> SC3CurrAverageDelay;
289
290       void UpdateAverageQueueLength1 (int band, double AverageQueueLength);
291       double GetAverageQueueLength1(int band);
292       std::vector<double> SC1CurrAverageLength1;
293       std::vector<double> SC2CurrAverageLength1;
294       std::vector<double> SC3CurrAverageLength1;
295
296       void UpdateAverageQueueLength2 (int band, double AverageQueueLength);
297       double GetAverageQueueLength2(int band);
298       std::vector<double> SC1CurrAverageLength2;
299       std::vector<double> SC2CurrAverageLength2;
300       std::vector<double> SC3CurrAverageLength2;
```

```
301
302     bool SetAverageQueueDelay1;
303     bool SetAverageQueueDelay2;
304     bool SetAverageQueueDelay3;
305
306
307 };
308
309 #endif // RDWQUEUE_H
310
```

```cpp
#include "RDWQueue.h"
#include "ns3/log.h"
#include "ns3/enum.h"
#include "ns3/uinteger.h"
#include "ns3/double.h"
#include "ns3/simulator.h"
#include "ns3/abort.h"
#include "ns3/random-variable-stream.h"
#include "Case1_conf.h"
#include "ns3/qos-tag.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("RDWQueue");
NS_OBJECT_ENSURE_REGISTERED(RDWQueue);

TypeId RDWQueue::GetTypeId(void)
{
    static TypeId tid = TypeId("ns3::RDWQueue")
                            .SetParent<Queue>()
                            .SetGroupName("Network")
                            .AddConstructor<RDWQueue>()
                            .AddAttribute("LinkBandwidth",
                                          "The RDWQueue link bandwidth",
                                          DataRateValue(DataRate("1.5Mbps")),
                                          MakeDataRateAccessor(&RDWQueue::m_linkBandwidth),
                                          MakeDataRateChecker())
                            .AddAttribute("LinkDelay",
                                          "The RDWQueue link delay",
                                          TimeValue(MilliSeconds(20)),
                                          MakeTimeAccessor(&RDWQueue::m_linkDelay),
                                          MakeTimeChecker());

    return tid;
}


// Constructor Function
RDWQueue::RDWQueue(int id) : Queue()
{
    linkID = id;
    // ctor
    sc1dropcount = 0;
    sc2dropcount = 0;
    sc3dropcount = 0;
    sc4dropcount = 0;

    sc1Hdropcount = 0;
    sc1Ldropcount = 0;
    sc1Bdropcount = 0;

    sc2Hdropcount = 0;
    sc2Ldropcount = 0;
    sc2Bdropcount = 0;

    sc3Hdropcount = 0;
    sc3Ldropcount = 0;
    sc3Bdropcount = 0;

```

```
60      SC1Weight = 4;
61      SC2Weight = 2;
62      SC3Weight = 1;
63
64      SC1BUCKETS = 10;
65      SC2BUCKETS = 10;
66      SC3BUCKETS = 10;
67
68      sc1qavg = 100;
69      sc2qavg = 100;
70      sc3qavg = 100;
71
72      REDSC1LD = 0;
73      REDSC2LD = 0;
74      REDSC3LD = 0;
75
76      sc1ByteLength = 0;
77      sc2ByteLength = 0;
78      sc3ByteLength = 0;
79
80
81      SC1Packets = 0;
82      SC2Packets = 0;
83      SC3Packets = 0;
84
85      SC1DepartPackets = 0;
86      SC2DepartPackets = 0;
87      SC3DepartPackets = 0;
88
89      SC4PKTsSC1 = 0;
90      SC4PKTsSC2 = 0;
91      SC4PKTsSC3 = 0;
92
93      SC1PrevUsage.push_back(sc1ByteLength);
94      SC2PrevUsage.push_back(sc2ByteLength);
95      SC3PrevUsage.push_back(sc3ByteLength);
96
97      SC1CurreUsage.push_back(sc1ByteLength);
98      SC2CurreUsage.push_back(sc2ByteLength);
99      SC3CurreUsage.push_back(sc3ByteLength);
100
101     SC1CurreDelay.push_back(0);
102     SC2CurreDelay.push_back(0);
103     SC3CurreDelay.push_back(0);
104
105
106     band = 0;
107     BYTEmode = 1;
108     CalRatePktType = 1;
109     PktDeqScPromo = 1;//1
110     DeQMode1CurrCls = 1;
111     currSecond = 0;
112     NumberOfPktsPerSec = 0;
113     NumberOfPktsPerK = 0;
114
115     for (double i = 10; i < Case1_conf::SIM_STOP_Sec; i +=1) {
116         Simulator::Schedule(Seconds(i), &RDWQueue::CalculateSchedulingRate, this);
117
118     }
119     for (double i = 10; i < Case1_conf::SIM_STOP_Sec; i += 1) {
120         Simulator::Schedule(Seconds(i), &RDWQueue::ShowStatisticReport, this);
```

```
121     }
122     for(double i = 1; i < Case1_conf::SIM_STOP_Sec; i += 1) {
123     Simulator::Schedule(Seconds(i), &RDWQueue::getLinkUtlization, this);
124     Simulator::Schedule(Seconds(i), &RDWQueue::getPktsPerSec, this);
125
126     }
127
128 }
129 // Destructor Function
130 RDWQueue::~RDWQueue()
131 {
132     // dtor
133 }
134 // DoEnqueue Function (Byte Mode)
135 bool RDWQueue::DoEnqueue(Ptr<Packet> p)
136 {
137     uint32_t band = Classify(p);
138     NS_LOG_FUNCTION(linkID << this << band);
139     double Ran;
140    if (BYTEmode) // byte mode
141     {
142         if (band == 1 && (p->GetUid() != Pk_ID || p->GetUid() == 0)) {
143             if ( sc1ByteLength < SC1BYTEBUFFER) {
144                 if ( sc1ByteLength > REDSC1THIGH) {
145                     Ran = GenRand();
146                     if (Ran < REDSC1HDROP) {
147                         Pk_ID = p->GetUid();
148                         Drop(p);
149                         sc1dropcount++;
150                         sc1Hdropcount++;
151                         NS_LOG_INFO(linkID << "  "
152                                            << "SC1 Hthreshold Drop"
153                                            << "  " << sc1Hdropcount);
154                     } else {
155                         Pk_ID = p->GetUid();
156                         m_packets1.push_back(p);
157                         sc1ByteLength += p->GetSize();
158                         SC1EnqueuTime[p->GetUid()] = Simulator::Now();
159                         SC1Packets++;
160                         NS_LOG_INFO("SC1Pkt_InsertTime"
161                                     << "  " << linkID << "  " << p->GetUid() << "    ↵
                                    " << p->GetSize() << "  "
162                                     << SC1EnqueuTime[p->GetUid()] << "tos" << band  ↵
                                    << "  " << SC1Packets);
163                     }
164                 } else if (sc1ByteLength > REDSC1TLOW) {
165                     Ran = GenRand();
166                     if (Ran < REDSC1LDROP) {
167                         Pk_ID = p->GetUid();
168                         Drop(p);
169                         sc1dropcount++;
170                         sc1Ldropcount++;
171                         NS_LOG_INFO(linkID << "  "
172                                            << "SC1 Lthreshold Drop"
173                                            << "  " << sc1Ldropcount);
174                     } else {
175                         Pk_ID = p->GetUid();
176                         m_packets1.push_back(p);
177                         sc1ByteLength += p->GetSize();
178                         SC1EnqueuTime[p->GetUid()] = Simulator::Now();
179                         SC1Packets++;
```

```
180                         NS_LOG_INFO("SC1Pkt_InsertTime"
181                                     << "  " << linkID << "  " << p->GetUid() << "  "
                                        " << p->GetSize() << "  "
182                                     << SC1EnqueuTime[p->GetUid()] << "  "
183                                     << "tos" << band << "  " << SC1Packets);
184                     }
185                 } else {
186                     Pk_ID = p->GetUid();
187                     m_packets1.push_back(p);
188                     sc1ByteLength += p->GetSize();
189                     SC1EnqueuTime[p->GetUid()] = Simulator::Now();
190                     SC1Packets++;
191                     NS_LOG_INFO("SC1Pkt_InsertTime"
192                                 << "  " << linkID << "  " << p->GetUid() << "  "
                                    << p->GetSize() << "  "
193                                 << SC1EnqueuTime[p->GetUid()] << "  "
194                                 << "tos" << band << "  " << SC1Packets);
195                 }
196             } else {
197                 Pk_ID = p->GetUid();
198                 Drop(p);
199                 sc1dropcount++;
200                 sc1Bdropcount++;
201                 NS_LOG_INFO(linkID << "  "
202                                    << "SC1 Buffer Drop"
203                                    << "  " << sc1Bdropcount);
204             }
205
206         } else if (band == 2 && (p->GetUid() != Pk_ID || p->GetUid() == 0)) {
207             if (sc2ByteLength < SC2BYTEBUFFER) {
208                 if (sc2ByteLength > REDSC2THIGH) {
209                     Ran = GenRand();
210                     if (Ran < REDSC2HDROP) {
211                         Pk_ID = p->GetUid();
212                         Drop(p);
213                         sc2dropcount++;
214                         sc2Hdropcount++;
215                         NS_LOG_INFO(linkID << "  "
216                                            << "SC2 Hthreshold Drop"
217                                            << "  " << sc2Hdropcount);
218                     } else {
219                         Pk_ID = p->GetUid();
220                         m_packets2.push_back(p);
221                         sc2ByteLength += p->GetSize();
222                         SC2EnqueuTime[p->GetUid()] = Simulator::Now();
223                         SC2Packets++;
224                         NS_LOG_INFO("SC2Pkt_InsertTime"
225                                     << "  " << linkID << "  " << p->GetUid() << "  "
                                        " << p->GetSize() << "  "
226                                     << SC2EnqueuTime[p->GetUid()] << "  "
227                                     << "tos" << band << "  " << SC2Packets);
228                     }
229                 } else if (sc2ByteLength  > REDSC2TLOW) {//
230                     Ran = GenRand();
231                     if (Ran < REDSC2LDROP) {
232                         Pk_ID = p->GetUid();
233                         Drop(p);
234                         sc2dropcount++;
235                         sc2Ldropcount++;
236                         NS_LOG_INFO(linkID << "  "
237                                            << "SC2 Lthreshold Drop"
```

```
238                                         << "   " << sc2Ldropcount);
239                     } else {
240                         Pk_ID = p->GetUid();
241                         m_packets2.push_back(p);
242                         sc2ByteLength += p->GetSize();
243                         SC2EnqueuTime[p->GetUid()] = Simulator::Now();
244                         SC2Packets++;
245                         NS_LOG_INFO("SC2Pkt_InsertTime"
246                                     << "   " << linkID << "   " << p->GetUid() << "    ↵
                                    " << p->GetSize() << "   "
247                                     << SC2EnqueuTime[p->GetUid()] << "   "
248                                     << "tos" << band << "   " << SC2Packets);
249                     }
250                 } else {
251                     Pk_ID = p->GetUid();
252                     m_packets2.push_back(p);
253                     sc2ByteLength += p->GetSize();
254                     SC2EnqueuTime[p->GetUid()] = Simulator::Now();
255                     SC2Packets++;
256                     NS_LOG_INFO("SC2Pkt_InsertTime"
257                                 << "   " << linkID << "   " << p->GetUid() << "   "    ↵
                                <<  p->GetSize() << "   "
258                                 << SC2EnqueuTime[p->GetUid()] << "   "
259                                 << "tos" << band << "   " << SC2Packets);
260                 }
261             } else {
262                 Pk_ID = p->GetUid();
263                 Drop(p);
264                 sc2dropcount++;
265                 sc2Bdropcount++;
266                 NS_LOG_INFO(linkID << "   "
267                                    << "SC2 Buffer Drop"
268                                    << "   " << sc2Bdropcount);
269             }
270         } else if (band == 3 && (p->GetUid() != Pk_ID || p->GetUid() == 0)) {
271             if (sc3ByteLength  < SC3BYTEBUFFER) {
272                 if (sc3ByteLength  > REDSC3THIGH) {
273                     Ran = GenRand();
274                     if (Ran < REDSC3HDROP) {
275                         Pk_ID = p->GetUid();
276                         Drop(p);
277                         sc3dropcount++;
278                         sc3Hdropcount++;
279                         NS_LOG_INFO(linkID << "   "
280                                            << "SC3 Hthreshold Drop"
281                                            << "   " << sc3Hdropcount);
282                     } else {
283                         Pk_ID = p->GetUid();
284                         m_packets3.push_back(p);
285                         sc3ByteLength += p->GetSize();
286                         SC3EnqueuTime[p->GetUid()] = Simulator::Now();
287                         SC3Packets++;
288                         NS_LOG_INFO("SC3Pkt_InsertTime"
289                                     << "   " << linkID << "   " << p->GetUid() << "    ↵
                                    " << p->GetSize() << "   "
290                                     << SC3EnqueuTime[p->GetUid()] << "   "
291                                     << "tos" << band << "   " << SC3Packets);
292                     }
293                 } else if (sc3ByteLength  > REDSC3TLOW) {
294                     Ran = GenRand();
295                     if (Ran < REDSC3LDROP) {
```

```
296                         Pk_ID = p->GetUid();
297                         Drop(p);
298                         sc3dropcount++;
299                         sc3Ldropcount++;
300                         NS_LOG_INFO(linkID << "  "
301                                             << "SC3 Lthreshold Drop"
302                                             << "  " << sc3Ldropcount);
303                     } else {
304                         Pk_ID = p->GetUid();
305                         m_packets3.push_back(p);
306                         sc3ByteLength += p->GetSize();
307                         SC3EnqueuTime[p->GetUid()] = Simulator::Now();
308                         SC3Packets++;
309                         NS_LOG_INFO("SC3Pkt_InsertTime"
310                                     << "  " << linkID << "  " << p->GetUid() << "  "
                                        " << p->GetSize() << "  "
311                                     << SC3EnqueuTime[p->GetUid()] << "  "
312                                     << "tos" << band << "  " << SC3Packets);
313                     }
314                 } else {
315                     Pk_ID = p->GetUid();
316                     m_packets3.push_back(p);
317                     sc3ByteLength += p->GetSize();
318                     SC3EnqueuTime[p->GetUid()] = Simulator::Now();
319                     SC3Packets++;
320                     NS_LOG_INFO("SC3Pkt_InsertTime"
321                                 << "  " << linkID << "  " << p->GetUid() << "  "
                                    << p->GetSize() << "  "
322                                 << SC3EnqueuTime[p->GetUid()] << "  "
323                                 << "tos" << band << "  " << SC3Packets);
324                 }
325             } else {
326                 Pk_ID = p->GetUid();
327                 Drop(p);
328                 sc3dropcount++;
329                 sc3Bdropcount++;
330                 NS_LOG_INFO(linkID << "  "
331                                    << "SC3 Buffer Drop"
332                                    << "  " << sc3Bdropcount);
333             }
334         } else if ( p->GetUid() != Pk_ID || p->GetUid() == 0) {
335             if (sc1ByteLength < (float)SC1BYTEBUFFER && sc1ByteLength <=
                sc2ByteLength &&
336                 sc1ByteLength <= sc3ByteLength ) {
337                 Pk_ID = p->GetUid();
338                 m_packets1.push_back(p);
339                 sc1ByteLength += p->GetSize();
340                 SC1EnqueuTime[p->GetUid()] = Simulator::Now();
341                 SC1Packets++;
342                 SC4PKTsSC1++;
343                 NS_LOG_INFO("SC4Pkt_Insert"
344                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                            p->GetSize() << "  "
345                             << "tos" << band << "  "
346                             << "SC4PKTs in SC1"
347                             << "  " << SC4PKTsSC1);
348                 NS_LOG_INFO("SC1Pkt_InsertTime"
349                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                            p->GetSize() << "  "
350                             << SC1EnqueuTime[p->GetUid()] << "  "
351                             << "tos" << band << "  " << SC1Packets);
```

```
352
353             } else if (sc2ByteLength < (float)SC2BYTEBUFFER && sc2ByteLength <=
                sc1ByteLength &&
354                         sc2ByteLength <= sc3ByteLength ) {
355                 Pk_ID = p->GetUid();
356                 m_packets2.push_back(p);
357                 sc2ByteLength += p->GetSize();
358                 SC2EnqueuTime[p->GetUid()] = Simulator::Now();
359                 SC2Packets++;
360                 SC4PKTsSC2++;
361                 NS_LOG_INFO("SC4Pkt_Insert"
362                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                            p->GetSize() << "  "
363                             << "tos" << band << "  "
364                             << "SC4PKTs in SC2"
365                             << "  " << SC4PKTsSC2);
366                 NS_LOG_INFO("SC2Pkt_InsertTime"
367                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                            p->GetSize() << "  "
368                             << SC2EnqueuTime[p->GetUid()] << "  "
369                             << "tos" << band << "  " << SC2Packets);
370
371             } else if (sc3ByteLength < (float)SC3BYTEBUFFER && sc3ByteLength  <=
                sc1ByteLength &&
372                         sc3ByteLength  <= sc2ByteLength) {
373                 Pk_ID = p->GetUid();
374                 m_packets3.push_back(p);
375                 sc3ByteLength += p->GetSize();
376                 SC3EnqueuTime[p->GetUid()] = Simulator::Now();
377                 SC3Packets++;
378                 SC4PKTsSC3++;
379                 NS_LOG_INFO("SC4Pkt_Insert"
380                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                            p->GetSize() << "  "
381                             << "tos" << band << "  "
382                             << "SC4PKTs in SC3"
383                             << "  " << SC4PKTsSC3);
384                 NS_LOG_INFO("SC3Pkt_InsertTime"
385                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                            p->GetSize() << "  "
386                             << SC3EnqueuTime[p->GetUid()] << "  "
387                             << "tos" << band << "  " << SC3Packets);
388
389             } else {
390                 Pk_ID = p->GetUid();
391                 Drop(p);
392                 sc4dropcount++;
393                 NS_LOG_INFO(linkID << "  "
394                                    << "SC4 Buffer Drop"
395                                    << "  " << sc4dropcount);
396                 return false;
397             }
398         }
399     }
400
401
402 // DoEnqueue Function (Packet Mode)
403 //packet mode, NOT CONSIDERED IN THE RESEARCH
404     else {
405         if (band == 1 && (p->GetUid() != Pk_ID || p->GetUid() == 0)) {
406             if (m_packets1.size() < SC1PKTBUFFER) {
```

```
                if (m_packets1.size() > REDSC1THIGHPKT) {
                    Ran = GenRand();
                    if (Ran < REDSC1HDROP) {
                        Pk_ID = p->GetUid();
                        Drop(p);
                        sc1dropcount++;
                        sc1Hdropcount++;
                        NS_LOG_INFO(linkID << "  "
                                           << "SC1 Hthreshold Drop"
                                           << "  " << sc1Hdropcount);
                    } else {
                        Pk_ID = p->GetUid();
                        m_packets1.push_back(p);
                        sc1ByteLength += p->GetSize();
                        SC1EnqueuTime[p->GetUid()] = Simulator::Now();
                        SC1Packets++;
                        NS_LOG_INFO("SC1Pkt_InsertTime"
                                    << "  " << linkID << "  " << p->GetUid() << "  "
                                    " << p->GetSize() << "  "
                                    << SC1EnqueuTime[p->GetUid()] << "tos" << band
                                    << "  " << SC1Packets);
                    }
                } else if (m_packets1.size() > REDSC1TLOWPKT) {
                    Ran = GenRand();
                    if (Ran < REDSC1LDROP) {
                        Pk_ID = p->GetUid();
                        Drop(p);
                        sc1dropcount++;
                        sc1Ldropcount++;
                        NS_LOG_INFO(linkID << "  "
                                           << "SC1 Lthreshold Drop"
                                           << "  " << sc1Ldropcount);
                    } else {
                        Pk_ID = p->GetUid();
                        m_packets1.push_back(p);
                        sc1ByteLength += p->GetSize();
                        SC1EnqueuTime[p->GetUid()] = Simulator::Now();
                        SC1Packets++;
                        NS_LOG_INFO("SC1Pkt_InsertTime"
                                    << "  " << linkID << "  " << p->GetUid() << "  "
                                    " << p->GetSize() << "  "
                                    << SC1EnqueuTime[p->GetUid()] << "  "
                                    << "tos" << band << "  " << SC1Packets);
                    }
                } else {
                    Pk_ID = p->GetUid();
                    m_packets1.push_back(p);
                    sc1ByteLength += p->GetSize();
                    SC1EnqueuTime[p->GetUid()] = Simulator::Now();
                    SC1Packets++;
                    NS_LOG_INFO("SC1Pkt_InsertTime"
                                << "  " << linkID << "  " << p->GetUid() << "  "
                                << p->GetSize() << "  "
                                << SC1EnqueuTime[p->GetUid()] << "  "
                                << "tos" << band << "  " << SC1Packets);
                }
            } else {
                Pk_ID = p->GetUid();
                Drop(p);
                sc1dropcount++;
                sc1Bdropcount++;
```

```
464                 NS_LOG_INFO(linkID << "  "
465                                    << "SC1 Buffer Drop"
466                                    << "  " << sc1Bdropcount);
467             }
468
469         } else if (band == 2 && (p->GetUid() != Pk_ID || p->GetUid() == 0)) {
470             if (m_packets2.size() < SC2PKTBUFFER) {
471                 if (m_packets2.size() > REDSC2THIGHPKT) {
472                     Ran = GenRand();
473                     if (Ran < REDSC2HDROP) {
474                         Pk_ID = p->GetUid();
475                         Drop(p);
476                         sc2dropcount++;
477                         sc2Hdropcount++;
478                         NS_LOG_INFO(linkID << "  "
479                                            << "SC2 Hthreshold Drop"
480                                            << "  " << sc2Hdropcount);
481                     } else {
482                         Pk_ID = p->GetUid();
483                         m_packets2.push_back(p);
484                         sc2ByteLength += p->GetSize();
485                         SC2EnqueuTime[p->GetUid()] = Simulator::Now();
486                         SC2Packets++;
487                         NS_LOG_INFO("SC2Pkt_InsertTime"
488                                     << "  " << linkID << "  " << p->GetUid() << "  "
                                    " << p->GetSize() << "  "
489                                     << SC2EnqueuTime[p->GetUid()] << "tos" << band
                                    << "  " << SC2Packets);
490                     }
491                 } else if (m_packets2.size() > REDSC2TLOWPKT) {
492                     Ran = GenRand();
493                     if (Ran < REDSC2LDROP) {
494                         Pk_ID = p->GetUid();
495                         Drop(p);
496                         sc2dropcount++;
497                         sc2Ldropcount++;
498                         NS_LOG_INFO(linkID << "  "
499                                            << "SC2 Lthreshold Drop"
500                                            << "  " << sc2Ldropcount);
501                     } else {
502                         Pk_ID = p->GetUid();
503                         m_packets2.push_back(p);
504                         sc2ByteLength += p->GetSize();
505                         SC2EnqueuTime[p->GetUid()] = Simulator::Now();
506                         SC2Packets++;
507                         NS_LOG_INFO("SC2Pkt_InsertTime"
508                                     << "  " << linkID << "  " << p->GetUid() << "  "
                                    " << p->GetSize() << "  "
509                                     << SC2EnqueuTime[p->GetUid()] << "  "
510                                     << "tos" << band << "  " << SC2Packets);
511                     }
512                 } else {
513                     Pk_ID = p->GetUid();
514                     m_packets2.push_back(p);
515                     sc2ByteLength += p->GetSize();
516                     SC2EnqueuTime[p->GetUid()] = Simulator::Now();
517                     SC2Packets++;
518                     NS_LOG_INFO("SC2Pkt_InsertTime"
519                                 << "  " << linkID << "  " << p->GetUid() << "  "
                                << p->GetSize() << "  "
520                                 << SC2EnqueuTime[p->GetUid()] << "  "
```

```
521                                 << "tos" << band << "  " << SC2Packets);
522                 }
523             } else {
524                 Pk_ID = p->GetUid();
525                 Drop(p);
526                 sc2dropcount++;
527                 sc2Bdropcount++;
528                 NS_LOG_INFO(linkID << "  "
529                                    << "SC2 Buffer Drop"
530                                    << "  " << sc2Bdropcount);
531             }
532         } else if (band == 3 && (p->GetUid() != Pk_ID || p->GetUid() == 0)) {
533             if (m_packets3.size() < SC3PKTBUFFER) {
534                 if (m_packets3.size() > REDSC3THIGHPKT) {
535                     Ran = GenRand();
536                     if (Ran < REDSC3HDROP) {
537                         Pk_ID = p->GetUid();
538                         Drop(p);
539                         sc3dropcount++;
540                         sc3Hdropcount++;
541                         NS_LOG_INFO(linkID << "  "
542                                            << "SC3 Hthreshold Drop"
543                                            << "  " << sc3Hdropcount);
544                     } else {
545                         Pk_ID = p->GetUid();
546                         m_packets3.push_back(p);
547                         sc3ByteLength += p->GetSize();
548                         SC3EnqueuTime[p->GetUid()] = Simulator::Now();
549                         SC3Packets++;
550                         NS_LOG_INFO("SC3Pkt_InsertTime"
551                                     << "  " << linkID << "  " << p->GetUid() << "  ↵
                                    " << p->GetSize() << "  "
552                                     << SC3EnqueuTime[p->GetUid()] << "tos" << band  ↵
                                    << "  " << SC3Packets);
553                     }
554                 } else if (m_packets3.size() > REDSC3TLOWPKT) {
555                     Ran = GenRand();
556                     if (Ran < REDSC3LDROP) {
557                         Pk_ID = p->GetUid();
558                         Drop(p);
559                         sc3dropcount++;
560                         sc3Ldropcount++;
561                         NS_LOG_INFO(linkID << "  "
562                                            << "SC3 Lthreshold Drop"
563                                            << "  " << sc3Ldropcount);
564                     } else {
565                         Pk_ID = p->GetUid();
566                         m_packets3.push_back(p);
567                         sc3ByteLength += p->GetSize();
568                         SC3EnqueuTime[p->GetUid()] = Simulator::Now();
569                         SC3Packets++;
570                         NS_LOG_INFO("SC3Pkt_InsertTime"
571                                     << "  " << linkID << "  " << p->GetUid() << "  ↵
                                    " << p->GetSize() << "  "
572                                     << SC3EnqueuTime[p->GetUid()] << "  "
573                                     << "tos" << band << "  " << SC3Packets);
574                     }
575                 } else {
576                     Pk_ID = p->GetUid();
577                     m_packets3.push_back(p);
578                     sc3ByteLength += p->GetSize();
```

```
579                     SC3EnqueuTime[p->GetUid()] = Simulator::Now();
580                     SC3Packets++;
581                     NS_LOG_INFO("SC3Pkt_InsertTime"
582                                 << "  " << linkID << "  " << p->GetUid() << "  "
                                    << p->GetSize() << "  "
583                                 << SC3EnqueuTime[p->GetUid()] << "  "
584                                 << "tos" << band << "  " << SC3Packets);
585                 }
586             } else {
587                 Pk_ID = p->GetUid();
588                 Drop(p);
589                 sc3dropcount++;
590                 sc3Bdropcount++;
591                 NS_LOG_INFO(linkID << "  "
592                                    << "SC3 Buffer Drop"
593                                    << "  " << sc3Bdropcount);
594             }
595         } else {
596             // update 141114
597             if (m_packets1.size() < (float)SC1PKTBUFFER && m_packets1.size() <=
                m_packets2.size() &&
598                 m_packets1.size() <= m_packets3.size() && (p->GetUid() != Pk_ID ||
                p->GetUid() == 0)) {
599                 Pk_ID = p->GetUid();
600                 m_packets1.push_back(p);
601                 sc1ByteLength += p->GetSize();
602                 SC1EnqueuTime[p->GetUid()] = Simulator::Now();
603                 SC1Packets++;
604                 SC4PKTsSC1++;
605                 NS_LOG_INFO("SC4Pkt_Insert"
606                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                            p->GetSize() << "  "
607                             << "tos" << band << "  "
608                             << "SC4PKTs in SC1"
609                             << "  " << SC4PKTsSC1);
610                 NS_LOG_INFO("SC1Pkt_InsertTime"
611                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                            p->GetSize() << "  "
612                             << SC1EnqueuTime[p->GetUid()] << "  "
613                             << "tos" << band << "  " << SC1Packets);
614
615             } else if (m_packets2.size() < (float)SC2PKTBUFFER &&
                m_packets2.size() <= m_packets1.size() &&
616                        m_packets2.size() <= m_packets3.size() && (p->GetUid() !=
                        Pk_ID || p->GetUid() == 0)) {
617                 Pk_ID = p->GetUid();
618                 m_packets2.push_back(p);
619                 sc2ByteLength += p->GetSize();
620                 SC2EnqueuTime[p->GetUid()] = Simulator::Now();
621                 SC2Packets++;
622                 SC4PKTsSC2++;
623                 NS_LOG_INFO("SC4Pkt_Insert"
624                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                            p->GetSize() << "  "
625                             << "tos" << band << "  "
626                             << "SC4PKTs in SC2"
627                             << "  " << SC4PKTsSC2);
628                 NS_LOG_INFO("SC2Pkt_InsertTime"
629                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                            p->GetSize() << "  "
630                             << SC2EnqueuTime[p->GetUid()] << "  "
```

```
631                             << "tos" << band << "  " << SC2Packets);
632
633             } else if (m_packets3.size() < (float)SC3PKTBUFFER &&
                m_packets3.size() <= m_packets1.size() &&
634                         m_packets3.size() <= m_packets2.size() && (p->GetUid() !=
                        Pk_ID || p->GetUid() == 0)) {
635                 Pk_ID = p->GetUid();
636                 m_packets3.push_back(p);
637                 sc3ByteLength += p->GetSize();
638                 SC3EnqueuTime[p->GetUid()] = Simulator::Now();
639                 SC3Packets++;
640                 SC4PKTsSC3++;
641                 NS_LOG_INFO("SC4Pkt_Insert"
642                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                            p->GetSize() << "  "
643                             << "tos" << band << "  "
644                             << "SC4PKTs in SC3"
645                             << "  " << SC4PKTsSC3);
646                 NS_LOG_INFO("SC3Pkt_InsertTime"
647                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                            p->GetSize() << "  "
648                             << SC3EnqueuTime[p->GetUid()] << "  "
649                             << "tos" << band << "  " << SC3Packets);
650
651             } else {
652                 Pk_ID = p->GetUid();
653                 Drop(p);
654                 sc4dropcount++;
655                 NS_LOG_INFO(linkID << "  "
656                                    << "SC4 Buffer Drop"
657                                    << "  " << sc4dropcount);
658
659                 return false;
660             }
661         }
662     }
663     return true;
664 }
665
666 // DoDequeue Function (weights (buckets) are Based on the number of Packets)
667
668 Ptr<Packet> RDWQueue::DoDequeue(void)
669 {
670     NS_LOG_INFO(linkID);
671     Ptr<Packet> p = NULL;
672     if (PktDeqScPromo == 1) { // class 1->2->3,1->2->3
673         if (DeQMode1CurrCls == 1 && m_packets1.size() > 0) {
674            if (SC1BUCKETS == 1 || m_packets1.size() <= 1) { // next class
675                 DeQMode1CurrCls = 2;
676                 SC1BUCKETS = SC1BUCKETSCPY + 1;
677             }
678             p = m_packets1.front();
679             SC1DequeuTime[p->GetUid()] = Simulator::Now();
680             m_packets1.pop_front();
681             sc1ByteLength -= p->GetSize();
682             SC1BUCKETS--;
683             band = Classify(p);
684
685             if (band != 0) {
686                 UpdateLinkUtlization(band, p->GetSize());
687                 UpdatePacketsPerSec(band);
```

```
688                 SC1DepartPackets++;
689                 NS_LOG_INFO("SC1Pkt_DepartureTime"
690                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                                p->GetSize() << "  "
691                             << SC1DequeuTime[p->GetUid()].GetSeconds() << "  "
692                             << "tos" << band << "  " << SC1DepartPackets);
693             }
694
695         } else if (DeQMode1CurrCls == 2 && m_packets2.size() > 0) {
696             if (SC2BUCKETS == 1 || m_packets2.size() <= 1) {
697                 DeQMode1CurrCls = 3;
698                 SC2BUCKETS = SC2BUCKETSCPY + 1;
699             }
700             p = m_packets2.front();
701             SC2DequeuTime[p->GetUid()] = Simulator::Now();
702             m_packets2.pop_front();
703             sc2ByteLength -= p->GetSize();
704             SC2BUCKETS--;
705             band = Classify(p);
706             if (band != 0) {
707                 UpdateLinkUtlization(band, p->GetSize());
708                 UpdatePacketsPerSec(band);
709                 SC2DepartPackets++;
710                 NS_LOG_INFO("SC2Pkt_DepartureTime"
711                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                                p->GetSize() << "  "
712                             << SC2DequeuTime[p->GetUid()].GetSeconds() << "  "
713                             << "tos" << band << "  " << SC2DepartPackets);
714             }
715         } else if (DeQMode1CurrCls == 3 && m_packets3.size() > 0) {
716             if (SC3BUCKETS == 1 || m_packets3.size() <= 1) {
717                 DeQMode1CurrCls = 1;
718                 SC3BUCKETS = SC3BUCKETSCPY + 1;
719             }
720             p = m_packets3.front();
721             SC3DequeuTime[p->GetUid()] = Simulator::Now();
722             m_packets3.pop_front();
723             sc3ByteLength -= p->GetSize();
724             SC3BUCKETS--;
725             band = Classify(p);
726             if (band != 0) {
727                 UpdateLinkUtlization(band, p->GetSize());
728                 UpdatePacketsPerSec(band);
729                 SC3DepartPackets++;
730                 NS_LOG_INFO("SC3Pkt_DepartureTime"
731                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                                p->GetSize() << "  "
732                             << (SC3DequeuTime[p->GetUid()]).GetSeconds() << "  "
733                             << "tos" << band << "  " << SC3DepartPackets);
734             }
735         }
736     } else { // class 1->2->3 with promote
737         if (SC1BUCKETS > 0) {
738             // sc1
739             NS_LOG_INFO(linkID << " SC1 Dequeue");
740             if (m_packets1.size()) {
741                 p = m_packets1.front();
742                 SC1DequeuTime[p->GetUid()] = Simulator::Now();
743                 m_packets1.pop_front();
744                 sc1ByteLength -= p->GetSize();
745                 SC1BUCKETS--;
```

```
746                 band = Classify(p);
747                 if (band != 0) {
748                     UpdateLinkUtlization(band, p->GetSize());
749                     UpdatePacketsPerSec(band);
750                     SC1DepartPackets++;
751                     NS_LOG_INFO("SC1Pkt_DepartureTime"
752                                 << "  " << linkID << "  " << p->GetUid() << "  "
                                    << p->GetSize() << "  "
753                                 << SC1DequeuTime[p->GetUid()].GetSeconds() << "  "
754                                 << "tos" << band << "  " << SC1DepartPackets);
755                 }
756
757             }
758         } else if (SC2BUCKETS > 0) {
759             // sc2
760             NS_LOG_INFO(linkID << " SC2 Dequeue");
761             if (m_packets2.size()) {
762                 p = m_packets2.front();
763                 SC2DequeuTime[p->GetUid()] = Simulator::Now();
764                 m_packets2.pop_front();
765                 sc2ByteLength -= p->GetSize();
766                 SC2BUCKETS--;
767                 band = Classify(p);
768                 if (band != 0) {
769                     UpdateLinkUtlization(band, p->GetSize());
770                     UpdatePacketsPerSec(band);
771                     SC2DepartPackets++;
772                     NS_LOG_INFO("SC2Pkt_DepartureTime"
773                                 << "  " << linkID << "  " << p->GetUid() << "  "
                                    << p->GetSize() << "  "
774                                 << SC2DequeuTime[p->GetUid()].GetSeconds() << "  "
775                                 << "tos" << band << "  " << SC2DepartPackets);
776                 }
777             }
778         }else if (SC3BUCKETS > 0) {
779             // sc3
780             NS_LOG_INFO(linkID << " SC3 Dequeue");
781             if (m_packets3.size()) {
782                 p = m_packets3.front();
783                 SC3DequeuTime[p->GetUid()] = Simulator::Now();
784                 m_packets3.pop_front();
785                 sc3ByteLength -= p->GetSize();
786                 SC3BUCKETS--;
787                 band = Classify(p);
788                 if (band != 0) {
789                     UpdateLinkUtlization(band, p->GetSize());
790                     UpdatePacketsPerSec(band);
791                     SC3DepartPackets++;
792                     NS_LOG_INFO("SC3Pkt_DepartureTime"
793                                 << "  " << linkID << "  " << p->GetUid() << "  "
                                    << p->GetSize() << "  "
794                                 << SC3DequeuTime[p->GetUid()].GetSeconds() << "  "
795                                 << "tos" << band << "  " << SC3DepartPackets);
796                 }
797             }
798         }
799         if (SC1BUCKETS == 0 && SC2BUCKETS == 0 && SC3BUCKETS == 0) {
800             SC1BUCKETS = SC1BUCKETSCPY;
801             SC2BUCKETS = SC2BUCKETSCPY;
802             SC3BUCKETS = SC3BUCKETSCPY;
803             NS_LOG_INFO ("All Zeros" << "  "<<Simulator::Now());
```

```
804         }
805     }
806
807
808 if (p == 0) { //NULL
809         if (m_packets1.size() >0) {
810             p = m_packets1.front();
811             SC1DequeuTime[p->GetUid()] = Simulator::Now();
812             m_packets1.pop_front();
813             sc1ByteLength -= p->GetSize();
814             SC1BUCKETS--;
815             band = Classify(p);
816             if (band != 0) {
817                 UpdateLinkUtlization(band, p->GetSize());
818                 UpdatePacketsPerSec(band);
819                 SC1DepartPackets++;
820                 NS_LOG_INFO(linkID << "  "
821                                    << "SC1 Passive Dequeue");
822                 NS_LOG_INFO("SC1Pkt_DepartureTime"
823                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                                p->GetSize() << "  "
824                             << SC1DequeuTime[p->GetUid()].GetSeconds() << "  "
825                             << "tos" << band << "  " << SC1DepartPackets);
826             }
827         }  else if (m_packets2.size() >0) {
828             p = m_packets2.front();
829             SC2DequeuTime[p->GetUid()] = Simulator::Now();
830             m_packets2.pop_front();
831             sc2ByteLength -= p->GetSize();
832             SC2BUCKETS--;
833             band = Classify(p);
834             if (band != 0) {
835                 UpdateLinkUtlization(band, p->GetSize());
836                 UpdatePacketsPerSec(band);
837                 SC2DepartPackets++;
838                 NS_LOG_INFO(linkID << "  "
839                                    << "SC2 Passive Dequeue");
840                 NS_LOG_INFO("SC2Pkt_DepartureTime"
841                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                                p->GetSize() << "  "
842                             << SC2DequeuTime[p->GetUid()].GetSeconds() << "  "
843                             << "tos" << band << "  " << SC2DepartPackets);
844             }
845         }
846         else if (m_packets3.size() >0 ) {
847             p = m_packets3.front();
848             SC3DequeuTime[p->GetUid()] = Simulator::Now();
849             m_packets3.pop_front();
850             sc3ByteLength -= p->GetSize();
851             SC3BUCKETS--;
852             band = Classify(p);
853             if (band != 0) {
854                 UpdateLinkUtlization(band, p->GetSize());
855                 UpdatePacketsPerSec(band);
856                 SC3DepartPackets++;
857                 NS_LOG_INFO(linkID << "  "
858                                    << "SC3 Passive Dequeue");
859                 NS_LOG_INFO("SC3Pkt_DepartureTime"
860                             << "  " << linkID << "  " << p->GetUid() << "  " <<
                                p->GetSize() << "  "
861                             << SC3DequeuTime[p->GetUid()].GetSeconds() << "  "
```

```
862                             << "tos" << band << "   " << SC3DepartPackets);
863                 }
864             }
865         }
866
867         if (currSecond != (int)Simulator::Now().GetSeconds()) {
868             NS_LOG_INFO(linkID << " NumberOfPktsPerSec " << currSecond << " is " <<
                NumberOfPktsPerSec);
869             NumberOfPktsPerSec = 0;
870             currSecond = (int)Simulator::Now().GetSeconds();
871             if ((int)currSecond % 20 == 0) {
872                 NS_LOG_INFO(linkID << " NumberOfPktsPerK " << currSecond << " is " <<
                    NumberOfPktsPerK);
873                 NumberOfPktsPerK = 0;
874             }
875         }
876
877         NumberOfPktsPerSec++;
878         NumberOfPktsPerK++;
879         return p;
880     }
881     // Updating (Calculating) the link utilization within an interval k Function.
882     void RDWQueue::UpdateLinkUtlization(uint32_t band, uint32_t pktsize)
883     {
884         if(band == 1) {
885         sc1pktsinsecond += pktsize;
886         } else if(band == 2) {
887         sc2pktsinsecond += pktsize;
888         } else if(band == 3) {
889         sc3pktsinsecond += pktsize;
890         }
891     }
892     // Get the link utilization within an interval k Function.
893     void RDWQueue::getLinkUtlization()
894     {
895         float a=0,b=0,c=0,d=0;
896
897         // for upstream links
898         if ((linkID == 134) || (linkID ==145) || // Scenairo 1 link capacity utilization
899             (linkID == 234) || (linkID ==245) ||  // Scenairo 2 link capacity utilization
900             (linkID == 323) || (linkID ==334) || (linkID == 356) || (linkID == 367))
                // Scenairo 3 link capacity utilization
901         {
902         a = (float)sc1pktsinsecond* 8 / Case1_conf::UpstreamLINKCAPACITY;
903         b = (float)sc2pktsinsecond * 8 / Case1_conf::UpstreamLINKCAPACITY;
904         c = (float)sc3pktsinsecond * 8 / Case1_conf::UpstreamLINKCAPACITY;
905         d = a + b + c;
906         std::cout << Simulator::Now() <<"  "<< "LinkUtilization in second for output
            port:"<< "   " << linkID <<"  "<< "Link Bandwidth (bits/sec):" <<"  "<<
            Case1_conf::LINKCAPACITY <<"  "<<"SC1:"<<"  "<< a <<" "<<"SC2:"<<"  "<< b <<"
            " <<"SC3:"<<"  "<< c <<" "<< "Total:" <<"  "<< d << std::endl;
907
908         }
909
910     // for downstream links
911     else if ( (linkID == 168) || (linkID == 189) || // Scenairo 1 link capacity
        utilization
912             (linkID ==268) || (linkID == 269) ||
913             (linkID == 2610) || (linkID ==2811) ||
914             (linkID ==2911)|| (linkID ==21011) || // Scenairo 2 link capacity utilization
915             (linkID ==31013)|| (linkID ==31113)||
```

```
916           (linkID ==31213) || (linkID ==3810) || (linkID ==3811) ||
917           (linkID ==3812) || (linkID ==3910) || (linkID ==3911)||
918           (linkID ==3912))     // Scenairo 3 link capacity utilization
919       {
920       a = (float)sc1pktsinsecond* 8 / Case1_conf::DownstreamLINKCAPACITY;
921       b = (float)sc2pktsinsecond * 8 / Case1_conf::DownstreamLINKCAPACITY;
922       c = (float)sc3pktsinsecond * 8 / Case1_conf::DownstreamLINKCAPACITY;
923       d = a + b + c;
924       std::cout << Simulator::Now() <<"  "<< "LinkUtilization in second for output
          port:"<< "  " << linkID <<"  "<< "Link Bandwidth (bits/sec):" <<"  "<<
          Case1_conf::LINKCAPACITY <<"  "<<"SC1:"<<"  "<< a <<" "<<"SC2:"<<"  "<< b <<"
          " <<"SC3:"<<"  "<< c <<" "<< "Total:" <<"  "<< d << std::endl;
925
926       }
927       // for domains links
928       else if ((linkID == 156) ||  // scenairo 1 between domains link capacity
          utilization
929                (linkID == 256) || // scenairo 2 between domains link capacity
930                (linkID == 348) || (linkID == 379))   // scenairo 3 between domains
                 link capacity utilization
931       {
932        a = (float)sc1pktsinsecond* 8 / Case1_conf::DomainsLINKCAPACITY;
933       b = (float)sc2pktsinsecond * 8 / Case1_conf::DomainsLINKCAPACITY;
934       c = (float)sc3pktsinsecond * 8 / Case1_conf::DomainsLINKCAPACITY;
935       d = a + b + c;
936       std::cout << Simulator::Now() <<"  "<< "DomainsLinkUtilization in second for
          output port:"<< "  " << linkID <<"  "<< "Link Bandwidth (bits/sec):" <<"  "<<
          Case1_conf::DomainsLINKCAPACITY <<"  "<<"SC1:"<<"  "<< a <<" "<<"SC2:"<<"  "<<
          b <<" " <<"SC3:"<<"  "<< c <<" "<< "Total:" <<"  "<< d << std::endl;
937
938       }
939       // for destinations links
940       else if ((linkID ==1910) ||
941                (linkID == 21112) || // scenairo 2 Destionation link capacity
942                (linkID == 31314))  // scenairo 3 Destionation link capacity
943       {
944       a = (float)sc1pktsinsecond* 8 / Case1_conf::DestinationLINKCAPACITY;
945       b = (float)sc2pktsinsecond * 8 / Case1_conf::DestinationLINKCAPACITY;
946       c = (float)sc3pktsinsecond * 8 / Case1_conf::DestinationLINKCAPACITY;
947       d = a + b + c;
948       std::cout << Simulator::Now() <<"  "<< "DestinationLinkUtilization in second
          for output port:"<< "  " << linkID <<"  "<< "Link Bandwidth (bits/sec):" <<"
          "<< Case1_conf::DestinationLINKCAPACITY <<"  "<<"SC1:"<<"  "<< a <<"
          "<<"SC2:"<<"  "<< b <<" " <<"SC3:"<<"  "<< c <<" "<< "Total:" <<"  "<< d <<
          std::endl;
949       }
950       sc1pktsinsecond = sc2pktsinsecond = sc3pktsinsecond = 0;
951   }
952
953   // Get service class queue size in byte function
954   uint32_t RDWQueue::GetQueueSizeBytes(uint32_t band)
955   {
956       if (band == 1) {
957           return SC1CurreUsage.back();
958       } else if (band == 2) {
959           return SC2CurreUsage.back();
960       } else if (band == 3) {
961           return SC3CurreUsage.back();
962       }
963       return 0;
964   }
```

```
965  // Classification Function
966  uint32_t RDWQueue::Classify(Ptr<const Packet> p)
967  {
968      uint32_t band = 0;
969      QosTag tosTag;
970      if (p->PeekPacketTag(tosTag)) {
971          band = (uint32_t)tosTag.GetTid();
972      }
973      return band;
974  }
975  // Generate Random number (GenRand) function
976  double RDWQueue::GenRand(void)
977  {
978      Ptr<ExponentialRandomVariable> x = CreateObject<ExponentialRandomVariable>();
979      x->SetAttribute("Mean", DoubleValue(1));
980      x->SetAttribute("Bound", DoubleValue(0.5));
981      double value = x->GetValue();
982      return value;
983  }
984  // Calculate the GCD value Function.
985  int calculateGCD(int a, int b)
986  {
987      if (b > a) {
988          int tmp = b;
989          b = a;
990          a = tmp;
991      }
992      if (a % b == 0) {
993          return b;
994      } else {
995          return calculateGCD(a % b, b);
996      }
997  }
998  // Calculate scheduling rates function for service classes in the edge and core routers for DiffServ domains
999  void RDWQueue::CalculateSchedulingRate(void)
1000 {
1001     float rates[3];
1002     float* res;
1003     if (BYTEmode) {
1004         if (linkID == 145 || linkID == 189 ||  //Scenairo1
1005            linkID == 245 || linkID == 2811 || linkID == 2911 || linkID == 21011
               //Scenairo 2 fixed weights queues
1006             || linkID == 334 || linkID == 367 || linkID == 31013 || linkID ==
               31113 || linkID == 31213) //sceniro 3 fixed weights queues
1007             {
1008             res = CalculateSchedulingRateFixed(rates); // For the core routers
1009             }
1010          else if (CalRatePktType == 1)
1011              {
1012              res = CalculateSchedulingRateByte(rates);  // For the edge routers
1013              }
1014         else
1015           {
1016            res = CalculateSchedulingRatePktType2(rates); // Not considered in research
1017           }
1018          }
1019     else {
1020          // packet mode, it is not used in the research.
1021        /* if (linkID == 45 || linkID == 89) {
```

```
1022                res = CalculateSchedulingRateFixed(rates);
1023
1024            } else if (CalRatePktType == 1) {
1025                res = CalculateSchedulingRatePktType1(rates);
1026            } else {
1027                res = CalculateSchedulingRatePktType2(rates);
1028            }*/
1029        }
1030        if (linkID == 268 || linkID == 269 || linkID == 2610 ||  // Scenairo 2
        (Dynamic and Average of average)
1031            linkID == 3810 || linkID == 3811 || linkID == 3812 || // Sceairo 3
            (Dynamic and Average of average)
1032            linkID == 3910 || linkID == 3911 || linkID == 3912) // Sceairo 3 (Dynamic
            and Average of average)
1033        {
1034        *res =0;
1035        *(res+1)=0;
1036        *(res+2)=0;
1037        }
1038        if ((linkID == 134) || (linkID ==145)|| (linkID ==156) ||
1039           (linkID ==168) || (linkID ==189) || (linkID ==1910) ||
1040           (linkID == 234) || (linkID ==245)||(linkID ==256) ||  // Scenairo 2 Fixed
           and Dynamic weights
1041           (linkID ==2811)|| (linkID ==2911)|| (linkID ==21011) || (linkID
           ==21112)||// Scenairo 2 Fixed and Dynamic weights
1042           (linkID ==323)||(linkID ==334)||(linkID ==356)||(linkID ==367)||(linkID
           ==348)||// Scenairo 3 Fixed and Dynamic weights
1043           (linkID ==379)||(linkID ==31013)||(linkID ==31113)||(linkID
           ==31213)||(linkID ==31314)) // Scenairo 3 Fixed and Dynamic weights
1044        {
1045        CalculateBucket(*res, *(res + 1), *(res + 2));
1046        }
1047    }
1048
1049    // Calculate fixed scheduling rates function for service classes in the core
        routers for DiffServ domains
1050    float* RDWQueue::CalculateSchedulingRateFixed(float rates[3])
1051    {
1052
1053        // calculate the estimated average queue length for service classes
1054        // queue length = (1-row)*Qlength(k-1) + row * Qlength(k)
1055        if (sc1ByteLength==0 || m_packets1.size() == 0)
1056            {
1057               SC1CurrUsage=0;
1058               SC1Usage.push_back(0);
1059               SC1CurreUsage.push_back(0);
1060               SC1PrevUsage.push_back(0);
1061            }
1062
1063            else
1064            {
1065        double SC1CurrUsage =
1066            (1 -
            Case1_conf::ROWFORAVGQLENGTH)*SC1PrevUsage[SC1PrevUsage.size()-2]+(Case1_con
            f::ROWFORAVGQLENGTH*sc1ByteLength);
1067        SC1Usage.push_back(sc1ByteLength);
1068        SC1CurreUsage.push_back(SC1CurrUsage);
1069        SC1PrevUsage.push_back(SC1CurrUsage);
1070        }
1071        NS_LOG_INFO("SC1 Average Queue Length"<<"  "<<linkID<<"  "<<"Instantenous SC1
        Length:"<<"  "<<SC1Usage.back()<<"  "<<"Previous SC1 Average Length:"<<"
```

```
      "<<SC1PrevUsage[SC1PrevUsage.size()-2]<<"  "<<"Average SC1 Length:"<<"
      "<<SC1CurreUsage.back());
1072  if (sc2ByteLength==0 || m_packets2.size() == 0)
1073      {
1074         SC2CurrUsage=0;
1075         SC2Usage.push_back(0);
1076         SC2CurreUsage.push_back(0);
1077         SC2PrevUsage.push_back(0);
1078      }
1079
1080      else
1081      {
1082  double SC2CurrUsage =
1083      (1 -
      Case1_conf::ROWFORAVGQLENGTH)*SC2PrevUsage[SC2PrevUsage.size()-2]+(Case1_con
      f::ROWFORAVGQLENGTH*sc2ByteLength);
1084  SC2Usage.push_back(sc2ByteLength);
1085  SC2CurreUsage.push_back(SC2CurrUsage);
1086  SC2PrevUsage.push_back(SC2CurrUsage);
1087  }
1088  NS_LOG_INFO("SC2 Average Queue Length"<<"  "<<linkID<<"  "<<"Instantenous SC2
      Length:"<<"  "<<SC2Usage.back()<<"  "<<"Previous SC2 Average Length:"<<"
      "<<SC2PrevUsage[SC2PrevUsage.size()-2]<<"  "<<"Average SC2 Length:"<<"
      "<<SC2CurreUsage.back());
1089
1090  if (sc3ByteLength==0 || m_packets3.size() == 0)
1091      {
1092         SC3CurrUsage=0;
1093         SC3Usage.push_back(0);
1094         SC3CurreUsage.push_back(0);
1095         SC3PrevUsage.push_back(0);
1096      }
1097
1098      else
1099      {
1100  double SC3CurrUsage =
1101      (1 -
      Case1_conf::ROWFORAVGQLENGTH)*SC3PrevUsage[SC3PrevUsage.size()-2]+(Case1_con
      f::ROWFORAVGQLENGTH*sc3ByteLength);
1102  SC3Usage.push_back(sc3ByteLength);
1103  SC3CurreUsage.push_back(SC3CurrUsage);
1104  SC3PrevUsage.push_back(SC3CurrUsage);
1105  }
1106  NS_LOG_INFO("SC3 Average Queue Length"<<"  "<<linkID<<"  "<<"Instantenous SC3
      Length:"<<"  "<<SC3Usage.back()<<"  "<<"Previous SC3 Average Length:"<<"
      "<<SC3PrevUsage[SC3PrevUsage.size()-2]<<"  "<<"Average SC3 Length:"<<"
      "<<SC3CurreUsage.back());
1107
1108  NS_LOG_INFO("Average Queue Length Report " << linkID << " "
      <<SC1CurreUsage.back()<<"  "<< SC2CurreUsage.back()<<"
      "<<SC3CurreUsage.back());
1109
1110
1111  // calculate the delay per packet in each class
1112
1113  // store all the packet delay for later use
1114  std::list<uint64_t> SC1CurrDelay;
1115  for (std::map<uint64_t, Time>::iterator iter = SC1DequeuTime.begin(); iter !=
      SC1DequeuTime.end(); iter++) {
1116      uint64_t delay = (iter->second).GetMilliSeconds() -
      SC1EnqueuTime[iter->first].GetMilliSeconds();
```

```
1117            if (delay < 3600000) {
1118                SC1CurrDelay.push_back(delay);
1119                SC1Delay.push_back(delay);
1120                SC1EnqueuTime.erase(iter->first);
1121                SC1DequeuTime.erase(iter->first);
1122            }
1123        }
1124        SC1DequeuTime.clear();
1125        // calculate the average delay for service class queues during this k
1126        float SC1AVGDELAY = 0;
1127        for (std::list<uint64_t>::iterator iter = SC1CurrDelay.begin(); iter !=
          SC1CurrDelay.end(); iter++) {
1128            SC1AVGDELAY += *iter;
1129        }
1130        SC1AVGDELAY = SC1AVGDELAY / SC1CurrDelay.size();
1131        if (std::isnan(SC1AVGDELAY))
1132        {
1133            SC1AVGDELAY=0;
1134        }
1135        SC1CurreDelay.push_back(SC1AVGDELAY);
1136        SC1CurrDelay.clear();
1137
1138        std::list<uint64_t> SC2CurrDelay;
1139        for (std::map<uint64_t, Time>::iterator iter = SC2DequeuTime.begin(); iter !=
          SC2DequeuTime.end(); iter++) {
1140            uint64_t delay = (iter->second).GetMilliSeconds() -
              SC2EnqueuTime[iter->first].GetMilliSeconds();
1141            if (delay < 3600000) {
1142                SC2CurrDelay.push_back(delay);
1143                SC2Delay.push_back(delay);
1144                SC2EnqueuTime.erase(iter->first);
1145                SC2DequeuTime.erase(iter->first);
1146            }
1147        }
1148        SC2DequeuTime.clear();
1149        float SC2AVGDELAY = 0;
1150        for (std::list<uint64_t>::iterator iter = SC2CurrDelay.begin(); iter !=
          SC2CurrDelay.end(); iter++) {
1151            SC2AVGDELAY += *iter;
1152        }
1153        SC2AVGDELAY = SC2AVGDELAY / SC2CurrDelay.size();
1154        if (std::isnan(SC2AVGDELAY))
1155        {
1156            SC2AVGDELAY=0;
1157
1158        }
1159        SC2CurreDelay.push_back(SC2AVGDELAY);
1160        SC2CurrDelay.clear();
1161
1162        std::list<uint64_t> SC3CurrDelay;
1163        for (std::map<uint64_t, Time>::iterator iter = SC3DequeuTime.begin(); iter !=
          SC3DequeuTime.end(); iter++) {
1164            uint64_t delay = (iter->second).GetMilliSeconds() -
              SC3EnqueuTime[iter->first].GetMilliSeconds();
1165            if (delay < 36000000) {
1166                SC3CurrDelay.push_back(delay);
1167                SC3Delay.push_back(delay);
1168                SC3EnqueuTime.erase(iter->first);
1169                SC3DequeuTime.erase(iter->first);
1170            }
1171        }
```

```
1172      SC3DequeuTime.clear();
1173      float SC3AVGDELAY = 0;
1174      for (std::list<uint64_t>::iterator iter = SC3CurrDelay.begin(); iter !=
          SC3CurrDelay.end(); iter++) {
1175          SC3AVGDELAY += *iter;
1176      }
1177      SC3AVGDELAY = SC3AVGDELAY / SC3CurrDelay.size();
1178      if (std::isnan(SC3AVGDELAY))
1179      {
1180          SC3AVGDELAY=0;
1181      }
1182      SC3CurreDelay.push_back(SC3AVGDELAY);
1183      SC3CurrDelay.clear();
1184
1185      NS_LOG_INFO("Average Queue Delay Report " << linkID << " "
          <<SC1CurreDelay.back()<<"  "<< SC2CurreDelay.back()<<"
          "<<SC3CurreDelay.back());
1186
1187      // Assigne fixed rates for service class queues at the core routers of the
          DiffServ domains.
1188      rates[0] = 5;//5
1189      rates[1] = 4;//4
1190      rates[2] = 3;//3
1191        NS_LOG_INFO("Queue Scheduling Rate Report " << linkID << " " <<rates[0]<<"
          "<<rates[1]<<"  "<<rates[2]);
1192
1193      return rates;
1194
1195  }
1196  // Calling the function of Calculating dynamic scheduling rates for service
    classes in the edge routers for DiffServ domains in bytes/sec.
1197
1198  float* RDWQueue::CalculateSchedulingRatePktType1(float rates[3])
1199  { // scheduling rate= length/delay , then gcd to bucket
1200      return CalculateSchedulingRateByte(rates);
1201  }
1202
1203  // Not Considered in the research
1204  /*float* RDWQueue::CalculateSchedulingRatePktType2(float rates[3])
1205  {
1206      rates = CalculateSchedulingRateByte(rates);
1207      // rate=length/delay, then bucket = rate/avg pkt_size
1208      float sc1pktlth = 0;
1209      for (std::deque<Ptr<Packet> >::iterator b = m_packets1.begin(); b !=
          m_packets1.end(); b++) {
1210          sc1pktlth += (*b)->GetSize();
1211      }
1212      sc1pktlth = sc1pktlth / m_packets1.size();
1213
1214      float sc2pktlth = 0;
1215      for (std::deque<Ptr<Packet> >::iterator b = m_packets2.begin(); b !=
          m_packets2.end(); b++) {
1216          sc2pktlth += (*b)->GetSize();
1217      }
1218      sc2pktlth = sc2pktlth / m_packets2.size();
1219
1220      float sc3pktlth = 0;
1221      for (std::deque<Ptr<Packet> >::iterator b = m_packets3.begin(); b !=
          m_packets3.end(); b++) {
1222          sc3pktlth += (*b)->GetSize();
1223      }
```

```
1224      sc3pktlth = sc3pktlth / m_packets3.size();
1225
1226      rates[0] = rates[0] / sc1pktlth;
1227      rates[1] = rates[1] / sc2pktlth;
1228      rates[2] = rates[2] / sc3pktlth;
1229      return rates;
1230  }*/
1231
1232  // Calculate dynamic scheduling rates function for service classes in the edge routers for DiffServ domains
1233
1234  float* RDWQueue::CalculateSchedulingRateByte(float rates[3])
1235  {
1236      // calculate the estimated average queue length for service classes
1237
1238      // queue length = (1-row)*Qlength(k-1) + row * Qlength(k)
1239          if (sc1ByteLength==0 || m_packets1.size() == 0)
1240          {
1241             SC1CurrUsage=0;
1242             SC1Usage.push_back(0);
1243             SC1CurreUsage.push_back(0);
1244             SC1PrevUsage.push_back(0);
1245             if (linkID == 268 || linkID == 269 || linkID == 2610 ||  // scenairo 2 Dynamic and Average of Average
1246                 linkID == 3810 || linkID == 3811 || linkID == 3812 || // scenairo 3 Dynamic and Average of Average
1247                 linkID == 3910 || linkID == 3911 || linkID == 3912) // scenairo 3 Dynamic and Average of Average
1248             {
1249               UpdateAverageQueueLength1(1,SC1CurreUsage.back());
1250               UpdateAverageQueueLength2(1,SC1CurreUsage.back());
1251             }
1252
1253          }
1254
1255          else
1256          {
1257          SC1CurrUsage =
1258          (1 - Case1_conf::ROWFORAVGQLENGTH)*SC1PrevUsage[SC1PrevUsage.size()-2]+(Case1_conf::ROWFORAVGQLENGTH*sc1ByteLength);
1259      SC1Usage.push_back(sc1ByteLength);  // display instantenous values.
1260      SC1CurreUsage.push_back(SC1CurrUsage);
1261      SC1PrevUsage.push_back(SC1CurrUsage);
1262      if (linkID == 268 || linkID == 269 || linkID == 2610 ||  // scenairo 2 Dynamic and Average of Average
1263          linkID == 3810 || linkID == 3811 || linkID == 3812 || // scenairo 3 Dynamic and Average of Average
1264          linkID == 3910 || linkID == 3911 || linkID == 3912) // scenairo 3 Dynamic and Average of Average
1265      {
1266      UpdateAverageQueueLength1(1,SC1CurreUsage.back());
1267      UpdateAverageQueueLength2(1,SC1CurreUsage.back());
1268      }
1269
1270
1271      }
1272      NS_LOG_INFO("SC1 Average Queue Length"<<"  "<<linkID<<"  "<<"Instantenous SC1 Length:"<<"  "<<SC1Usage.back()<<"  "<<"Previous SC1 Average Length:"<<"  "<<SC1PrevUsage[SC1PrevUsage.size()-2]<<"  "<<"Average SC1 Length:"<<"
```

```
      "<<SC1CurreUsage.back());
1273
1274      if (sc2ByteLength==0 || m_packets2.size() == 0)
1275          {
1276             SC2CurrUsage=0;
1277             SC2Usage.push_back(0);
1278             SC2CurreUsage.push_back(0);
1279             SC2PrevUsage.push_back(0);
1280             if (linkID == 268 || linkID == 269 || linkID == 2610 || // scenairo 2
                 Dynamic and Average of Average
1281                 linkID == 3810 || linkID == 3811 || linkID == 3812 || // scenairo 3
                    Dynamic and Average of Average
1282                 linkID == 3910 || linkID == 3911 || linkID == 3912) // scenairo 3
                    Dynamic and Average of Average
1283             {
1284              UpdateAverageQueueLength1(2,SC2CurreUsage.back());
1285              UpdateAverageQueueLength2(2,SC2CurreUsage.back());
1286             }
1287
1288          }
1289
1290          else
1291      {
1292          SC2CurrUsage =
1293          (1 -                                                                    
          Case1_conf::ROWFORAVGQLENGTH)*SC2PrevUsage[SC2PrevUsage.size()-2]+(Case1_con
          f::ROWFORAVGQLENGTH*sc2ByteLength);
1294      SC2Usage.push_back(sc2ByteLength);
1295      SC2CurreUsage.push_back(SC2CurrUsage);
1296      SC2PrevUsage.push_back(SC2CurrUsage);
1297      if (linkID == 268 || linkID == 269 || linkID == 2610 || // scenairo 2 Dynamic
      and Average of Average
1298          linkID == 3810 || linkID == 3811 || linkID == 3812 || // scenairo 3
          Dynamic and Average of Average
1299          linkID == 3910 || linkID == 3911 || linkID == 3912) // scenairo 3 Dynamic
          and Average of Average
1300      {
1301      UpdateAverageQueueLength1(2,SC2CurreUsage.back());
1302      UpdateAverageQueueLength2(2,SC2CurreUsage.back());
1303      }
1304
1305      }
1306     NS_LOG_INFO("SC2 Average Queue Length"<<"  "<<linkID<<"  "<<"Instantenous SC2
      Length:"<<"  "<<SC2Usage.back()<<"  "<<"Previous SC2 Average Length:"<<"
      "<<SC2PrevUsage[SC2PrevUsage.size()-2]<<"  "<<"Average SC2 Length:"<<"
      "<<SC2CurreUsage.back());
1307
1308    if (sc3ByteLength==0 || m_packets3.size() == 0)
1309          {
1310            SC3CurrUsage=0;
1311             SC3Usage.push_back(0);
1312             SC3CurreUsage.push_back(0);
1313             SC3PrevUsage.push_back(0);
1314             if (linkID == 268 || linkID == 269 || linkID == 2610 || // scenairo 2
                 Dynamic and Average of Average
1315                 linkID == 3810 || linkID == 3811 || linkID == 3812 || // scenairo 3
                    Dynamic and Average of Average
1316                 linkID == 3910 || linkID == 3911 || linkID == 3912) // scenairo 3
                    Dynamic and Average of Average
1317             {
1318               UpdateAverageQueueLength1(3,SC3CurreUsage.back());
```

```
1319              UpdateAverageQueueLength2(3,SC3CurreUsage.back());
1320            }
1321
1322          }
1323
1324          else
1325      {
1326          SC3CurrUsage =
1327          (1 -                                                                    ↵
          Case1_conf::ROWFORAVGQLENGTH)*SC3PrevUsage[SC3PrevUsage.size()-2]+(Case1_con↵
          f::ROWFORAVGQLENGTH*sc3ByteLength);
1328      SC3Usage.push_back(sc3ByteLength);
1329      SC3CurreUsage.push_back(SC3CurrUsage);
1330      SC3PrevUsage.push_back(SC3CurrUsage);
1331
1332      // For routers with multi out ports
1333      if (linkID == 268 || linkID == 269 || linkID == 2610 || // scenairo 2 Dynamic   ↵
      and Average of Average
1334          linkID == 3810 || linkID == 3811 || linkID == 3812 || // scenairo 3         ↵
          Dynamic and Average of Average
1335          linkID == 3910 || linkID == 3911 || linkID == 3912) // scenairo 3 Dynamic   ↵
          and Average of Average
1336      {
1337      UpdateAverageQueueLength1(3,SC3CurreUsage.back());
1338      UpdateAverageQueueLength2(3,SC3CurreUsage.back());
1339      }
1340
1341      // To display the average queue length
1342  }
1343      NS_LOG_INFO("SC3 Average Queue Length"<<"  "<<linkID<<"  "<<"Instantenous SC3   ↵
      Length:"<<"  "<<SC3Usage.back()<<"  "<<"Previous SC3 Average Length:"<<"         ↵
      "<<SC3PrevUsage[SC3PrevUsage.size()-2]<<"  "<<"Average SC3 Length:"<<"           ↵
      "<<SC3CurreUsage.back());
1344
1345      NS_LOG_INFO("Average Queue Length Report " << linkID << " "                   ↵
      <<SC1CurreUsage.back()<<"  "<< SC2CurreUsage.back()<<"                          ↵
      "<<SC3CurreUsage.back());
1346
1347
1348      // calculate the delay per packet in each class
1349
1350      // store all the packet delay for later use
1351      std::list<uint64_t> SC1CurrDelay;
1352      for (std::map<uint64_t, Time>::iterator iter = SC1DequeuTime.begin(); iter !=  ↵
      SC1DequeuTime.end(); iter++) {
1353          uint64_t delay = (iter->second).GetMilliSeconds() -                       ↵
          SC1EnqueuTime[iter->first].GetMilliSeconds();
1354          if (delay < 3600000 ) {
1355              SC1CurrDelay.push_back(delay);
1356              SC1Delay.push_back(delay);
1357              SC1EnqueuTime.erase(iter->first);
1358              SC1DequeuTime.erase(iter->first);
1359          }
1360
1361      }
1362      SC1DequeuTime.clear();
1363      // calculate the average delay for each service class queue during this k
1364      float SC1AVGDELAY = 0;
1365      for (std::list<uint64_t>::iterator iter = SC1CurrDelay.begin(); iter !=       ↵
      SC1CurrDelay.end(); iter++) {
1366          SC1AVGDELAY += *iter;
```

```
1367        }
1368        SC1AVGDELAY = SC1AVGDELAY / SC1CurrDelay.size();
1369        if (std::isnan(SC1AVGDELAY) || std::isinf(SC1AVGDELAY))
1370        {
1371            SC1AVGDELAY=0;
1372        }
1373        SC1CurreDelay.push_back(SC1AVGDELAY); // to store average queue delay value
1374
1375        // For routers with multi out ports
1376        if (linkID == 268 || linkID == 269 || linkID == 2610 ||
1377            linkID == 3810 || linkID == 3811 || linkID == 3812 ||
1378            linkID == 3910 || linkID == 3911 || linkID == 3912)
1379        {
1380        UpdateAverageQueueDelay(1,SC1CurreDelay.back());
1381        }
1382        SC1CurrDelay.clear();
1383
1384
1385        std::list<uint64_t> SC2CurrDelay;
1386        for (std::map<uint64_t, Time>::iterator iter = SC2DequeuTime.begin(); iter !=
          SC2DequeuTime.end(); iter++) {
1387            uint64_t delay = (iter->second).GetMilliSeconds() -
              SC2EnqueuTime[iter->first].GetMilliSeconds();
1388            if (delay < 3600000 ) {
1389                SC2CurrDelay.push_back(delay);
1390                SC2Delay.push_back(delay);
1391                SC2EnqueuTime.erase(iter->first);
1392                SC2DequeuTime.erase(iter->first);
1393            }
1394
1395        }
1396        SC2DequeuTime.clear();
1397        float SC2AVGDELAY = 0;
1398        for (std::list<uint64_t>::iterator iter = SC2CurrDelay.begin(); iter !=
          SC2CurrDelay.end(); iter++) {
1399            SC2AVGDELAY += *iter;
1400        }
1401        SC2AVGDELAY = SC2AVGDELAY / SC2CurrDelay.size();
1402        if (std::isnan(SC2AVGDELAY)|| std::isinf(SC2AVGDELAY))
1403        {
1404            SC2AVGDELAY=0;
1405        }
1406        SC2CurreDelay.push_back(SC2AVGDELAY);
1407
1408           // For routers with multi out ports
1409        if (linkID == 268 || linkID == 269 || linkID == 2610 ||
1410            linkID == 3810 || linkID == 3811 || linkID == 3812 ||
1411            linkID == 3910 || linkID == 3911 || linkID == 3912)
1412        {
1413        UpdateAverageQueueDelay(2,SC2CurreDelay.back());
1414        }
1415        SC2CurrDelay.clear();
1416
1417
1418        std::list<uint64_t> SC3CurrDelay;
1419        for (std::map<uint64_t, Time>::iterator iter = SC3DequeuTime.begin(); iter !=
          SC3DequeuTime.end(); iter++) {
1420            uint64_t delay = (iter->second).GetMilliSeconds() -
              SC3EnqueuTime[iter->first].GetMilliSeconds();
1421            if (delay < 36000000 ) {
1422                SC3CurrDelay.push_back(delay);
```

```
1423                SC3Delay.push_back(delay);
1424                SC3EnqueuTime.erase(iter->first);
1425                SC3DequeuTime.erase(iter->first);
1426            }
1427        }
1428        SC3DequeuTime.clear();
1429        float SC3AVGDELAY = 0;
1430        for (std::list<uint64_t>::iterator iter = SC3CurrDelay.begin(); iter !=
        SC3CurrDelay.end(); iter++) {
1431            SC3AVGDELAY += *iter;
1432        }
1433        SC3AVGDELAY = SC3AVGDELAY / SC3CurrDelay.size();
1434        if (std::isnan(SC3AVGDELAY) || std::isinf(SC3AVGDELAY))
1435        {
1436            SC3AVGDELAY=0;
1437        }
1438        SC3CurreDelay.push_back(SC3AVGDELAY); // to store the average queue delay value
1439
1440            // For routers with multi out ports
1441        if (linkID == 268 || linkID == 269 || linkID == 2610 ||
1442            linkID == 3810 || linkID == 3811 || linkID == 3812 ||
1443            linkID == 3910 || linkID == 3911 || linkID == 3912)
1444        {
1445        UpdateAverageQueueDelay(3,SC3CurreDelay.back());
1446        }
1447        SC3CurrDelay.clear();
1448
1449         NS_LOG_INFO("Average Queue Delay Report " << linkID << " "
         <<SC1CurreDelay.back()<<"  "<< SC2CurreDelay.back()<<"
         "<<SC3CurreDelay.back());
1450
1451
1452    // Calculating dynamic scheduling rates for service classes within an interval k
1453    if (SC1CurreDelay.back() !=0 && SC2CurreDelay.back() !=0 &&
    SC3CurreDelay.back()!=0 && SC1CurreUsage.back()!=0 && SC2CurreUsage.back()!=0 &&
    SC3CurreUsage.back()!=0)  //1
1454        {
1455        SC1ScheRate = SC1CurreUsage.back()/(SC1CurreDelay.back()/1000) ;
1456
1457        // SC 2 scheduling Rate = 2 * r1 * q2 / q1
1458        SC2ScheRate = 0.5 * SC1ScheRate * SC2CurreUsage.back()/SC1CurreUsage.back();
1459
1460        // SC 3 scheduling Rate = 4* r1 * q3 / q1
1461        SC3ScheRate =  0.25 *SC1ScheRate * SC3CurreUsage.back()/SC1CurreUsage.back();
1462
1463
1464        }
1465        else if (SC1CurreDelay.back()==0 || SC1CurreUsage.back()==0)  //2
1466        {
1467         SC1ScheRate=0;
1468         SC2ScheRate=SC2CurreUsage.back()/(SC2CurreDelay.back()/1000);
1469         SC3ScheRate=0.5*SC2ScheRate*(SC3CurreUsage.back()/SC2CurreUsage.back());
1470
1471        }
1472        else if ((SC1CurreDelay.back()==0 || SC1CurreUsage.back()==0) &&
        (SC2CurreUsage.back()==0 ||SC2CurreDelay.back()==0 ))   //3
1473        {
1474            SC1ScheRate=0;
1475            SC2ScheRate=0;
1476            SC3ScheRate = SC3CurreUsage.back()/(SC3CurreDelay.back()/1000);
1477
```

```
1478        }
1479        else if ((SC1CurreDelay.back()==0 || SC1CurreUsage.back()==0) &&
            (SC3CurreDelay.back()==0|| SC3CurreUsage.back()==0))   //4
1480        {
1481            SC1ScheRate=0;
1482            SC2ScheRate = SC2CurreUsage.back()/(SC2CurreDelay.back()/1000);
1483            SC3ScheRate=0;
1484
1485        }
1486        else if ((SC2CurreDelay.back()==0 || SC2CurreUsage.back()==0) &&
            (SC3CurreDelay.back() ==0  || SC3CurreUsage.back()==0))  //5
1487        {
1488            SC1ScheRate = SC1CurreUsage.back()/(SC1CurreDelay.back()/1000);
1489            SC2ScheRate=0;
1490            SC3ScheRate=0;
1491
1492        }
1493        else if (SC2CurreDelay.back()==0 || SC2CurreUsage.back()==0 )  //6
1494        {
1495         SC1ScheRate = SC1CurreUsage.back() / (SC1CurreDelay.back()/1000);
1496         SC2ScheRate=0;
1497         SC3ScheRate = 0.5 * SC1ScheRate * SC3CurreUsage.back()/SC1CurreUsage.back();
1498
1499
1500        }
1501        else if (SC3CurreDelay.back()==0 || SC3CurreUsage.back()==0)  //7
1502        {
1503         SC1ScheRate = SC1CurreUsage.back()/(SC1CurreDelay.back()/1000);
1504         SC2ScheRate = 0.5*SC1ScheRate*(SC2CurreUsage.back()/SC1CurreUsage.back());
1505         SC3ScheRate=0;
1506
1507        }
1508        else if ((SC1CurreDelay.back()==0 && SC1CurreUsage.back()==0) &&
            (SC2CurreDelay.back()==0 && SC2CurreUsage.back()==0) &&
            (SC3CurreDelay.back()==0 && SC3CurreUsage.back()==0))   //8
1509        {
1510            SC1ScheRate =0;
1511            SC2ScheRate =0;
1512            SC3ScheRate =0;
1513
1514        }
1515
1516        if (std::isnan(SC1ScheRate) || std::isinf(SC1ScheRate))
1517            {
1518            SC1ScheRate=0;
1519            }
1520            else if (std::isnan(SC2ScheRate) || std::isinf(SC2ScheRate))
1521             {
1522            SC2ScheRate=0;
1523             }
1524             else if (std::isnan(SC3ScheRate) || std::isinf(SC3ScheRate))
1525             {
1526            SC3ScheRate=0;
1527             }
1528        NS_LOG_INFO("Queue Scheduling Rate Report "<< linkID << " " <<SC1ScheRate<<"
            "<< SC2ScheRate<<"  "<<SC3ScheRate);
1529
1530        NS_LOG_INFO("CalculateSchedulingRate " << linkID << " " << SC1ScheRate << " "
            << SC2ScheRate << " " << SC3ScheRate
1531                                            << " " << SC1CurreUsage.back()<< " " <<
                                                SC2CurreUsage.back()<< " "
```

```
                                        <<SC3CurreUsage.back());
1532
1533
1534  if ((linkID == 134) || (linkID ==156) || (linkID == 168) || (linkID == 1910) ||
      //scenairo 1 Dynamic scheduling rate calculation
1535          (linkID == 234)|| (linkID == 256)|| (linkID ==21112)||  //scenairo 2
              Dynamic scheduling rate calculation
1536          (linkID == 323) || (linkID ==356) || (linkID == 348)||  // scenairo 3
              Dynamic scheduling rate calculation
1537          (linkID == 379) || (linkID == 31314)) // scenairo 3 Dynamic scheduling
              rate calculation
1538      {
1539      rates[0] = SC1ScheRate;
1540      rates[1] = SC2ScheRate;
1541      rates[2] = SC3ScheRate;
1542      }
1543
1544      // For routers with multi out ports
1545
1546  else if ((linkID == 268) ||  (linkID == 269) || (linkID ==2610)||
1547                (linkID == 3810) ||  (linkID == 3811) || (linkID ==3812)||
1548                (linkID == 3910) ||  (linkID == 3911) || (linkID ==3912))  //
                  scenairo 2 set zero to sceduling rate
1549      {
1550      rates[0] = 0;
1551      rates[1] = 0;
1552      rates[2] = 0;
1553      }
1554
1555      return rates;
1556  }
1557
1558  // Calculate Bucket function
1559  bool RDWQueue::CalculateBucket(float SC1ScheRate, float SC2ScheRate, float
      SC3ScheRate)
1560  {   float r1,r2,r3;
1561      NS_LOG_INFO("NEW SCHEDULE RATE VALUE " << linkID << " " << SC1ScheRate << " "
      << SC2ScheRate << " " << SC3ScheRate);
1562      if (SC1ScheRate==0 && SC2ScheRate==0 && SC3ScheRate==0)
1563      {
1564           r1=r2=r3=0;
1565      }
1566      else
1567      {
1568       r1 = SC1ScheRate / (SC1ScheRate + SC2ScheRate + SC3ScheRate);
1569       r2 = SC2ScheRate / (SC1ScheRate + SC2ScheRate + SC3ScheRate);
1570       r3 = SC3ScheRate / (SC1ScheRate + SC2ScheRate + SC3ScheRate);
1571
1572      SC1ScheRate = r1 / (r1 + r2 + r3);
1573      SC2ScheRate = r2 / (r1 + r2 + r3);
1574      SC3ScheRate = r3 / (r1 + r2 + r3);
1575      }
1576      NS_LOG_INFO("NEW SCHEDULE RATE NORMALIZED VALUE " << linkID << " " <<
      SC1ScheRate << " " << SC2ScheRate << " "
1577                                                        << SC3ScheRate);
1578      // gcd
1579      int i1 = (int)(SC1ScheRate * 100);
1580      int i2 = (int)(SC2ScheRate * 100);
1581      int i3 = (int)(SC3ScheRate * 100);
1582      int gcd = 0;
1583
```

```
1584
1585  double sc1pktlth=0, sc2pktlth=0, sc3pktlth=0;
1586
1587  // For upstream links
1588  if ((linkID == 134) ||  //Scenairo 1 dynamic calculate weight according to Link capacity
1589      (linkID == 234) ||  //Scenairo 2 dynamic calculate weight according to Link capacity
1590      (linkID == 323) || (linkID ==356) ) //Scenairo 3 dynamic calculate weight according to Link capacity // Scenairo 3 dynamic calculate weight according to Link capacity
1591      {
1592
1593  if (i1 == 0) {
1594          if (i2 == 0) {
1595              if (i3 == 0) {
1596                  // 0,0,0   //1
1597                  gcd = 1;
1598                  SC1BUCKETS =0;
1599                  SC2BUCKETS =0;
1600                  SC3BUCKETS =0;
1601                  sc1pktlth=0;
1602                  sc2pktlth=0;
1603                  sc3pktlth=0;
1604                  std::cout<<"stat 1"<<" "<<linkID<<" "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<" "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<" "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<" "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1605
1606              } else if (i3 !=0){
1607                  // 0,0,1    //2
1608                  gcd = i3;
1609                  sc1pktlth=0;
1610                  sc2pktlth=0;
1611                   for (std::deque<Ptr<Packet> >::iterator c = m_packets3.begin(); c != m_packets3.end(); c++) {
1612                   sc3pktlth += (*c)->GetSize();
1613                      }
1614                   sc3pktlth = sc3pktlth / m_packets3.size();
1615                   SC1BUCKETS =0;
1616                   SC2BUCKETS =0;
1617                   SC3BUCKETS =( Case1_conf::UpstreamLINKCAPACITY)/(8*sc3pktlth);//350
1618
1619                   std::cout<<"stat 2"<<" "<<linkID<<" "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<" "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<" "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<" "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1620
1621              }
1622          } else {
1623              if (i3 == 0) {
1624                  // 0,1,0   //3
1625                  gcd = i2;
1626                  sc1pktlth=0;
1627                  for (std::deque<Ptr<Packet> >::iterator b = m_packets2.begin(); b != m_packets2.end(); b++) {
1628                   sc2pktlth += (*b)->GetSize();
1629                      }
1630                   sc2pktlth = sc2pktlth / m_packets2.size();
```

```
1631                    sc3pktlth =0;
1632                    SC1BUCKETS =0;
1633                   SC2BUCKETS =(Case1_conf::UpstreamLINKCAPACITY)/(8*sc2pktlth);//1400
1634                    SC3BUCKETS =0;
1635                     std::cout<<"stat 3"<<" "<<linkID<<"                                    ↵
                        "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"                   ↵
                        "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"                      ↵
                        "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"           ↵
                        "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1636
1637                } else {
1638                    // 0,1,1   //4
1639                    gcd = calculateGCD(i2, i3);
1640                    sc1pktlth=0;
1641                       for (std::deque<Ptr<Packet> >::iterator b = m_packets2.begin();  ↵
                        b != m_packets2.end(); b++) {
1642                     sc2pktlth += (*b)->GetSize();
1643                        }
1644                     sc2pktlth = sc2pktlth / m_packets2.size();
1645                     for (std::deque<Ptr<Packet> >::iterator c = m_packets3.begin(); c  ↵
                     != m_packets3.end(); c++) {
1646                     sc3pktlth += (*c)->GetSize();
1647                        }
1648                     sc3pktlth = sc3pktlth / m_packets3.size();
1649                     SC1BUCKETS =0;
1650                     SC2BUCKETS =((i2/gcd) *                                            ↵
                     Case1_conf::UpstreamLINKCAPACITY)/(100*8*sc2pktlth);//1400
1651                     SC3BUCKETS =((i3/gcd) *                                            ↵
                     Case1_conf::UpstreamLINKCAPACITY)/(100*8*sc3pktlth);//350
1652                     std::cout<<"stat 4"<<" "<<linkID<<"                                ↵
                     "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"               ↵
                     "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"                  ↵
                     "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"       ↵
                     "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1653
1654                }
1655            }    //i1==1
1656        } else {
1657            if (i2 == 0) {
1658                if (i3 == 0) {
1659                    // 1,0,0    //5
1660                    std::cout<<"stat 5"<<" "<<linkID<<"                                ↵
                    "<<Simulator::Now()/1000000000<<"  "<<"i1"<<"  "<<i1<<"i2"<<"      ↵
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1661                    gcd = i1;
1662                    for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a  ↵
                    != m_packets1.end(); a++) {
1663                     sc1pktlth += (*a)->GetSize();
1664                        }
1665                     sc1pktlth = sc1pktlth / m_packets1.size();
1666                     sc2pktlth=0;
1667                     sc3pktlth=0;
1668                     SC1BUCKETS =(Case1_conf::UpstreamLINKCAPACITY)/(8*sc1pktlth);
1669
1670                     SC2BUCKETS =0;
1671                     SC3BUCKETS =0;
1672                      std::cout<<"stat 5"<<" "<<linkID<<"                               ↵
                      "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"              ↵
                      "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"                 ↵
                      "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"      ↵
                      "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
```

```
1673
1674
1675            } else {
1676                // 1,0,1     //6
1677                gcd = calculateGCD(i1, i3);
1678                for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a
                    != m_packets1.end(); a++) {
1679                 sc1pktlth += (*a)->GetSize();
1680                    }
1681                 sc1pktlth = sc1pktlth / m_packets1.size();
1682                 sc2pktlth=0;
1683                 for (std::deque<Ptr<Packet> >::iterator c = m_packets3.begin(); c
                    != m_packets3.end(); c++) {
1684                 sc3pktlth += (*c)->GetSize();
1685                    }
1686                 sc3pktlth = sc3pktlth / m_packets3.size();
1687                 SC1BUCKETS =((i1/gcd) *
                    Case1_conf::UpstreamLINKCAPACITY)/(100*8*sc1pktlth);
1688                 SC2BUCKETS =0;
1689                 SC3BUCKETS =((i3/gcd) *
                    Case1_conf::UpstreamLINKCAPACITY)/(100*8*sc3pktlth);
1690                  std::cout<<"stat 6"<<" "<<linkID<<"
                     "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                     "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                     "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                     "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1691
1692            }
1693        } else {
1694            if (i3 == 0) {
1695                // 1,1,0    //7
1696                gcd = calculateGCD(i1, i2);
1697                for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a
                    != m_packets1.end(); a++) {
1698                 sc1pktlth += (*a)->GetSize();
1699                    }
1700                 sc1pktlth = sc1pktlth / m_packets1.size();
1701                     for (std::deque<Ptr<Packet> >::iterator b =
                        m_packets2.begin(); b != m_packets2.end(); b++) {
1702                 sc2pktlth += (*b)->GetSize();
1703                    }
1704                 sc2pktlth = sc2pktlth / m_packets2.size();
1705                 sc3pktlth =0;
1706                 SC1BUCKETS =((i1/gcd) *
                    Case1_conf::UpstreamLINKCAPACITY)/(100*8*sc1pktlth);
1707                 SC2BUCKETS =((i2/gcd) *
                    Case1_conf::UpstreamLINKCAPACITY)/(100*8*sc2pktlth);
1708                 SC3BUCKETS =0;
1709                std::cout<<"stat 7"<<" "<<linkID<<"
                    "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                    "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                    "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1710
1711
1712            } else {
1713                // 1,1,1   //8
1714                gcd = calculateGCD(calculateGCD(i1, i2), i3);
1715                for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a
                    != m_packets1.end(); a++) {
1716                 sc1pktlth += (*a)->GetSize();
```

```
1717                    }
1718                  sc1pktlth = sc1pktlth / m_packets1.size();
1719                      for (std::deque<Ptr<Packet> >::iterator b =
                          m_packets2.begin(); b != m_packets2.end(); b++) {
1720                  sc2pktlth += (*b)->GetSize();
1721                     }
1722                  sc2pktlth = sc2pktlth / m_packets2.size();
1723                  for (std::deque<Ptr<Packet> >::iterator c = m_packets3.begin(); c
                  != m_packets3.end(); c++) {
1724                  sc3pktlth += (*c)->GetSize();
1725                     }
1726                  sc3pktlth = sc3pktlth / m_packets3.size();
1727                  SC1BUCKETS =((i1/gcd) *
                  Case1_conf::UpstreamLINKCAPACITY)/(100*8*sc1pktlth);
1728                  SC2BUCKETS =((i2/gcd) *
                  Case1_conf::UpstreamLINKCAPACITY)/(100*8*sc2pktlth);
1729                  SC3BUCKETS =((i3/gcd) *
                  Case1_conf::UpstreamLINKCAPACITY)/(100*8*sc3pktlth);
1730                 std::cout<<"stat 8"<<" "<<linkID<<"
                  "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                  "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                  "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                  "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1731
1732              }
1733          }
1734
1735      }
1736          NS_LOG_INFO("NEW SCHEDULE RATE BUCKETS " << linkID << " " <<"
              "<<Case1_conf::UpstreamLINKCAPACITY<<"  "<< gcd << " " << SC1BUCKETS << "
              " << SC2BUCKETS << " "
1737                                              << SC3BUCKETS);
1738  }
1739
1740
1741  //for downstream links
1742  else if ( (linkID ==168)  || //Scenairo 1 dynamic calculate weight according to
      Link capacity
1743       (linkID ==268) || (linkID == 269) || (linkID == 2610) || //Scenairo 2 dynamic
       calculate weight according to Link capacity
1744      (linkID == 3810) || (linkID ==3811 ) || //Scenairo 3 dynamic calculate weight
      according to Link capacity
1745      (linkID ==3812) || (linkID ==3910) || (linkID ==3911)|| (linkID ==3912)) //
      Scenairo 3 dynamic calculate weight according to Link capacity
1746      {
1747
1748  if (i1 == 0) {
1749          if (i2 == 0) {
1750              if (i3 == 0) {
1751                  // 0,0,0   //1
1752                  gcd = 1;
1753                  SC1BUCKETS =0;
1754                  SC2BUCKETS =0;
1755                  SC3BUCKETS =0;
1756                  sc1pktlth=0;
1757                  sc2pktlth=0;
1758                  sc3pktlth=0;
1759                  std::cout<<"stat 1"<<" "<<linkID<<"
                  "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                  "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                  "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
```

```
                "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1760
1761            } else if (i3 !=0){
1762                // 0,0,1    //2
1763                gcd = i3;
1764                sc1pktlth=0;
1765                sc2pktlth=0;
1766                 for (std::deque<Ptr<Packet> >::iterator c = m_packets3.begin(); c
                    != m_packets3.end(); c++) {
1767                 sc3pktlth += (*c)->GetSize();
1768                    }
1769                 sc3pktlth = sc3pktlth / m_packets3.size();
1770                 SC1BUCKETS =0;
1771                 SC2BUCKETS =0;
1772                 SC3BUCKETS =( Case1_conf::DownstreamLINKCAPACITY)/(8*sc3pktlth);
1773
1774                 std::cout<<"stat 2"<<" "<<linkID<<"
                    "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                    "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                    "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1775
1776            }
1777        } else {
1778            if (i3 == 0) {
1779                // 0,1,0   //3
1780                gcd = i2;
1781                sc1pktlth=0;
1782                for (std::deque<Ptr<Packet> >::iterator b = m_packets2.begin(); b
                    != m_packets2.end(); b++) {
1783                 sc2pktlth += (*b)->GetSize();
1784                    }
1785                 sc2pktlth = sc2pktlth / m_packets2.size();
1786                 sc3pktlth =0;
1787                 SC1BUCKETS =0;
1788                SC2BUCKETS =(Case1_conf::DownstreamLINKCAPACITY)/(8*sc2pktlth);
1789
1790                 SC3BUCKETS =0;
1791                  std::cout<<"stat 3"<<" "<<linkID<<"
                    "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                    "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                    "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1792
1793            } else {
1794                // 0,1,1   //4
1795                gcd = calculateGCD(i2, i3);
1796                sc1pktlth=0;
1797                   for (std::deque<Ptr<Packet> >::iterator b = m_packets2.begin();
                    b != m_packets2.end(); b++) {
1798                 sc2pktlth += (*b)->GetSize();
1799                    }
1800                 sc2pktlth = sc2pktlth / m_packets2.size();
1801                 for (std::deque<Ptr<Packet> >::iterator c = m_packets3.begin(); c
                    != m_packets3.end(); c++) {
1802                 sc3pktlth += (*c)->GetSize();
1803                    }
1804                 sc3pktlth = sc3pktlth / m_packets3.size();
1805                 SC1BUCKETS =0;
1806                 SC2BUCKETS =((i2/gcd) *
                    Case1_conf::DownstreamLINKCAPACITY)/(100*8*sc2pktlth);
```

```
1807                SC3BUCKETS =((i3/gcd) *                                          ⏎
                    Case1_conf::DownstreamLINKCAPACITY)/(100*8*sc3pktlth);
1808                std::cout<<"stat 4"<<" "<<linkID<<"                              ⏎
                    "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"             ⏎
                    "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"                ⏎
                    "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"     ⏎
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1809
1810            }
1811        }    //i1==1
1812    } else {
1813        if (i2 == 0) {
1814            if (i3 == 0) {
1815                // 1,0,0    //5
1816                std::cout<<"stat 5"<<" "<<linkID<<"                              ⏎
                    "<<Simulator::Now()/1000000000<<"  "<<"i1"<<"  "<<i1<<"i2"<<"    ⏎
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1817                gcd = i1;
1818                for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a  ⏎
                    != m_packets1.end(); a++) {
1819                 sc1pktlth += (*a)->GetSize();
1820                    }
1821                 sc1pktlth = sc1pktlth / m_packets1.size();
1822                 sc2pktlth=0;
1823                 sc3pktlth=0;
1824                  SC1BUCKETS =(Case1_conf::DownstreamLINKCAPACITY)/(8*sc1pktlth);
1825
1826                 SC2BUCKETS =0;
1827                 SC3BUCKETS =0;
1828                  std::cout<<"stat 5"<<" "<<linkID<<"                            ⏎
                     "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"           ⏎
                     "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"              ⏎
                     "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"   ⏎
                     "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1829
1830
1831            } else {
1832                // 1,0,1     //6
1833                gcd = calculateGCD(i1, i3);
1834                for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a  ⏎
                    != m_packets1.end(); a++) {
1835                 sc1pktlth += (*a)->GetSize();
1836                    }
1837                 sc1pktlth = sc1pktlth / m_packets1.size();
1838                 sc2pktlth=0;
1839                 for (std::deque<Ptr<Packet> >::iterator c = m_packets3.begin(); c  ⏎
                    != m_packets3.end(); c++) {
1840                 sc3pktlth += (*c)->GetSize();
1841                    }
1842                 sc3pktlth = sc3pktlth / m_packets3.size();
1843                 SC1BUCKETS =((i1/gcd) *                                        ⏎
                    Case1_conf::DownstreamLINKCAPACITY)/(100*8*sc1pktlth);
1844                 SC2BUCKETS =0;
1845                 SC3BUCKETS =((i3/gcd) *                                        ⏎
                    Case1_conf::DownstreamLINKCAPACITY)/(100*8*sc3pktlth);
1846                  std::cout<<"stat 6"<<" "<<linkID<<"                           ⏎
                     "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"          ⏎
                     "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"             ⏎
                     "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"  ⏎
                     "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1847
```

```
1848            }
1849        } else {
1850            if (i3 == 0) {
1851                // 1,1,0    //7
1852                gcd = calculateGCD(i1, i2);
1853                for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a
                    != m_packets1.end(); a++) {
1854                 sc1pktlth += (*a)->GetSize();
1855                    }
1856                 sc1pktlth = sc1pktlth / m_packets1.size();
1857                     for (std::deque<Ptr<Packet> >::iterator b =
                        m_packets2.begin(); b != m_packets2.end(); b++) {
1858                 sc2pktlth += (*b)->GetSize();
1859                    }
1860                 sc2pktlth = sc2pktlth / m_packets2.size();
1861                 sc3pktlth =0;
1862                 SC1BUCKETS =((i1/gcd) *
                    Case1_conf::DownstreamLINKCAPACITY)/(100*8*sc1pktlth);
1863                 SC2BUCKETS =((i2/gcd) *
                    Case1_conf::DownstreamLINKCAPACITY)/(100*8*sc2pktlth);
1864                 SC3BUCKETS =0;
1865                std::cout<<"stat 7"<<" "<<linkID<<"
                    "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                    "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                    "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1866
1867
1868            } else {
1869                // 1,1,1   //8
1870                gcd = calculateGCD(calculateGCD(i1, i2), i3);
1871                for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a
                    != m_packets1.end(); a++) {
1872                 sc1pktlth += (*a)->GetSize();
1873                    }
1874                 sc1pktlth = sc1pktlth / m_packets1.size();
1875                     for (std::deque<Ptr<Packet> >::iterator b =
                        m_packets2.begin(); b != m_packets2.end(); b++) {
1876                 sc2pktlth += (*b)->GetSize();
1877                    }
1878                 sc2pktlth = sc2pktlth / m_packets2.size();
1879                 for (std::deque<Ptr<Packet> >::iterator c = m_packets3.begin(); c
                    != m_packets3.end(); c++) {
1880                 sc3pktlth += (*c)->GetSize();
1881                    }
1882                 sc3pktlth = sc3pktlth / m_packets3.size();
1883                 SC1BUCKETS =((i1/gcd) *
                    Case1_conf::DownstreamLINKCAPACITY)/(100*8*sc1pktlth);
1884                 SC2BUCKETS =((i2/gcd) *
                    Case1_conf::DownstreamLINKCAPACITY)/(100*8*sc2pktlth);
1885                 SC3BUCKETS =((i3/gcd) *
                    Case1_conf::DownstreamLINKCAPACITY)/(100*8*sc3pktlth);
1886                std::cout<<"stat 8"<<" "<<linkID<<"
                    "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                    "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                    "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1887
1888            }
1889        }
1890
```

```
1891      }
1892          NS_LOG_INFO("NEW SCHEDULE RATE BUCKETS " << linkID << " " <<"
          "<<Case1_conf::DownstreamLINKCAPACITY<<"  "<< gcd << " " << SC1BUCKETS <<
          " " << SC2BUCKETS << " "
1893                                                << SC3BUCKETS);
1894
1895  }
1896
1897  //for domains links
1898   else if (linkID == 156 || // Scnairo 1 calculte weight of link between upstream
      and downstream domains
1899              linkID == 256 || // Scnairo 2 calculte weight of link between upstream
              and downstream domains
1900              linkID == 348 || linkID == 379 ){   //Scenairo 3 calculte weight of
              link between upstream and downstream domains
1901      if (i1 == 0) {
1902          if (i2 == 0) {
1903              if (i3 == 0) {
1904                  // 0,0,0   //1
1905                  gcd = 1;
1906                  sc1pktlth=0;
1907                  sc2pktlth=0;
1908                  sc3pktlth=0;
1909                  SC1BUCKETS =0;
1910                  SC2BUCKETS =0;
1911                  SC3BUCKETS =0;
1912                  std::cout<<"stat 1"<<" "<<linkID<<"
                    "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                    "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                    "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1913
1914              } else {
1915                  // 0,0,1    //2
1916                  gcd = i3;
1917                  sc1pktlth=0;
1918                  sc2pktlth=0;
1919                  for (std::deque<Ptr<Packet> >::iterator c = m_packets3.begin(); c
                    != m_packets3.end(); c++) {
1920                   sc3pktlth += (*c)->GetSize();
1921                      }
1922                   sc3pktlth = sc3pktlth / m_packets3.size();
1923                   SC1BUCKETS =0;
1924                   SC2BUCKETS =0;
1925                   SC3BUCKETS =(Case1_conf::DomainsLINKCAPACITY)/(8*sc3pktlth);
1926                   std::cout<<"stat 2"<<" "<<linkID<<"
                     "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                     "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                     "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                     "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1927
1928              }
1929          } else {
1930              if (i3 == 0) {
1931                  // 0,1,0   //3
1932                  gcd = i2;
1933                  sc1pktlth=0;
1934                  for (std::deque<Ptr<Packet> >::iterator b = m_packets2.begin(); b
                    != m_packets2.end(); b++) {
1935                   sc2pktlth += (*b)->GetSize();
1936                      }
```

```
1937                    sc2pktlth = sc2pktlth / m_packets2.size();
1938                    sc3pktlth =0;
1939                    SC1BUCKETS =0;
1940                    SC2BUCKETS =(Case1_conf::DomainsLINKCAPACITY)/(8*sc2pktlth);
1941                    SC3BUCKETS =0;
1942                    std::cout<<"stat 3"<<" "<<linkID<<"                                    ↵
                        "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"                   ↵
                        "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"                      ↵
                        "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"           ↵
                        "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1943
1944                }  else {
1945                    // 0,1,1   //4
1946                    gcd = calculateGCD(i2, i3);
1947                    sc1pktlth=0;
1948                       for (std::deque<Ptr<Packet> >::iterator b = m_packets2.begin();  ↵
                          b != m_packets2.end(); b++) {
1949                     sc2pktlth += (*b)->GetSize();
1950                        }
1951                     sc2pktlth = sc2pktlth / m_packets2.size();
1952                         for (std::deque<Ptr<Packet> >::iterator c =                    ↵
                         m_packets3.begin(); c != m_packets3.end(); c++) {
1953                     sc3pktlth += (*c)->GetSize();
1954                        }
1955                     sc3pktlth = sc3pktlth / m_packets3.size();
1956                     SC1BUCKETS =0;
1957                     SC2BUCKETS =((i2/gcd) *                                           ↵
                         Case1_conf::DomainsLINKCAPACITY)/(100*8*sc2pktlth);
1958                     SC3BUCKETS =((i3/gcd) *                                           ↵
                         Case1_conf::DomainsLINKCAPACITY)/(100*8*sc3pktlth);
1959                     std::cout<<"stat 4"<<" "<<linkID<<"                               ↵
                         "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"              ↵
                         "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"                 ↵
                         "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"      ↵
                         "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1960
1961                }
1962            }    //i1==1
1963        } else {
1964            if (i2 == 0) {
1965                if (i3 == 0) {
1966                    // 1,0,0    //5
1967                    gcd = i1;
1968                    for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a  ↵
                        != m_packets1.end(); a++) {
1969                     sc1pktlth += (*a)->GetSize();
1970                        }
1971                     sc1pktlth = sc1pktlth / m_packets1.size();
1972                     sc2pktlth=0;
1973                     sc3pktlth=0;
1974                     SC1BUCKETS =( Case1_conf::DomainsLINKCAPACITY)/(8*sc1pktlth);
1975                     SC2BUCKETS =0;
1976                     SC3BUCKETS =0;
1977                     std::cout<<"stat 5"<<" "<<linkID<<"                               ↵
                         "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"              ↵
                         "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"                 ↵
                         "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"      ↵
                         "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1978
1979
1980                }  else {
```

```
1981                // 1,0,1     //6
1982                gcd = calculateGCD(i1, i3);
1983                for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a
                    != m_packets1.end(); a++) {
1984                 sc1pktlth += (*a)->GetSize();
1985                    }
1986                 sc1pktlth = sc1pktlth / m_packets1.size();
1987                 sc2pktlth=0;
1988                 for (std::deque<Ptr<Packet> >::iterator c = m_packets3.begin(); c
                    != m_packets3.end(); c++) {
1989                 sc3pktlth += (*c)->GetSize();
1990                    }
1991                 sc3pktlth = sc3pktlth / m_packets3.size();
1992                 SC1BUCKETS =((i1/gcd) *
                    Case1_conf::DomainsLINKCAPACITY)/(100*8*sc1pktlth);
1993                 SC2BUCKETS =0;
1994                 SC3BUCKETS =((i3/gcd) *
                    Case1_conf::DomainsLINKCAPACITY)/(100*8*sc3pktlth);
1995                 std::cout<<"stat 6"<<" "<<linkID<<"
                    "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                    "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                    "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
1996
1997            }
1998        } else {
1999            if (i3 == 0) {
2000                // 1,1,0    //7
2001                gcd = calculateGCD(i1, i2);
2002                for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a
                    != m_packets1.end(); a++) {
2003                 sc1pktlth += (*a)->GetSize();
2004                    }
2005                 sc1pktlth = sc1pktlth / m_packets1.size();
2006                     for (std::deque<Ptr<Packet> >::iterator b =
                        m_packets2.begin(); b != m_packets2.end(); b++) {
2007                 sc2pktlth += (*b)->GetSize();
2008                    }
2009                 sc2pktlth = sc2pktlth / m_packets2.size();
2010                 sc3pktlth =0;
2011                 SC1BUCKETS =((i1/gcd) *
                    Case1_conf::DomainsLINKCAPACITY)/(100*8*sc1pktlth);
2012                 SC2BUCKETS =((i2/gcd) *
                    Case1_conf::DomainsLINKCAPACITY)/(100*8*sc2pktlth);
2013                 SC3BUCKETS =0;
2014                 std::cout<<"stat 7"<<" "<<linkID<<"
                    "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                    "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                    "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
2015
2016
2017
2018            } else {
2019                // 1,1,1   //8
2020                gcd = calculateGCD(calculateGCD(i1, i2), i3);
2021                for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a
                    != m_packets1.end(); a++) {
2022                 sc1pktlth += (*a)->GetSize();
2023                    }
2024                 sc1pktlth = sc1pktlth / m_packets1.size();
```

```
2025                    for (std::deque<Ptr<Packet> >::iterator b =
                        m_packets2.begin(); b != m_packets2.end(); b++) {
2026                sc2pktlth += (*b)->GetSize();
2027                   }
2028                sc2pktlth = sc2pktlth / m_packets2.size();
2029                    for (std::deque<Ptr<Packet> >::iterator c =
                        m_packets3.begin(); c != m_packets3.end(); c++) {
2030                sc3pktlth += (*c)->GetSize();
2031                   }
2032                sc3pktlth = sc3pktlth / m_packets3.size();
2033                SC1BUCKETS =((i1/gcd) *
                    Case1_conf::DomainsLINKCAPACITY)/(100*8*sc1pktlth);
2034                SC2BUCKETS =((i2/gcd) *
                    Case1_conf::DomainsLINKCAPACITY)/(100*8*sc2pktlth);
2035                SC3BUCKETS =((i3/gcd) *
                    Case1_conf::DomainsLINKCAPACITY)/(100*8*sc3pktlth);
2036               std::cout<<"stat 8"<<" "<<linkID<<"
                    "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                    "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                    "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
2037
2038            }
2039        }
2040    }
2041
2042    NS_LOG_INFO("NEW SCHEDULE RATE BUCKETS " << linkID << " " <<"
    "<<Case1_conf::DomainsLINKCAPACITY<<"  "<< gcd << " " << SC1BUCKETS << " " <<
    SC2BUCKETS << " "
2043                                            << SC3BUCKETS);
2044
2045
2046 }
2047 // For destinations links
2048 else if (linkID ==1910 || linkID == 21112 ||// Scenairo 2 weight of destination link
2049          linkID == 31314) { // Scenairo 3 weight of destination link
2050     if (i1 == 0) {
2051         if (i2 == 0) {
2052             if (i3 == 0) {
2053                 // 0,0,0   //1
2054                 gcd = 1;
2055                 sc1pktlth=0;
2056                 sc2pktlth=0;
2057                 sc3pktlth=0;
2058                 SC1BUCKETS =0;
2059                 SC2BUCKETS =0;
2060                 SC3BUCKETS =0;
2061                 std::cout<<"stat 1"<<" "<<linkID<<"
                    "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                    "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                    "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
2062
2063             } else {
2064                 // 0,0,1    //2
2065                 gcd = i3;
2066                 sc1pktlth=0;
2067                 sc2pktlth=0;
2068                 for (std::deque<Ptr<Packet> >::iterator c = m_packets3.begin(); c
                    != m_packets3.end(); c++) {
2069                  sc3pktlth += (*c)->GetSize();
```

```
2070                    }
2071                 sc3pktlth = sc3pktlth / m_packets3.size();
2072                 SC1BUCKETS =0;
2073                 SC2BUCKETS =0;
2074                 SC3BUCKETS =(Case1_conf::DestinationLINKCAPACITY)/(8*sc3pktlth);
2075                 std::cout<<"stat 2"<<" "<<linkID<<"
                    "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                    "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                    "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                    "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
2076
2077             }
2078         } else {
2079             if (i3 == 0) {
2080                 // 0,1,0   //3
2081                 gcd = i2;
2082                 sc1pktlth=0;
2083                 for (std::deque<Ptr<Packet> >::iterator b = m_packets2.begin(); b
                    != m_packets2.end(); b++) {
2084                  sc2pktlth += (*b)->GetSize();
2085                     }
2086                  sc2pktlth = sc2pktlth / m_packets2.size();
2087                  sc3pktlth =0;
2088                  SC1BUCKETS =0;
2089                  SC2BUCKETS =(Case1_conf::DestinationLINKCAPACITY)/(8*sc2pktlth);
2090                  SC3BUCKETS =0;
2091                  std::cout<<"stat 3"<<" "<<linkID<<"
                     "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                     "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                     "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                     "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
2092
2093             } else {
2094                 // 0,1,1   //4
2095                 gcd = calculateGCD(i2, i3);
2096                 sc1pktlth=0;
2097                    for (std::deque<Ptr<Packet> >::iterator b = m_packets2.begin();
                       b != m_packets2.end(); b++) {
2098                  sc2pktlth += (*b)->GetSize();
2099                     }
2100                  sc2pktlth = sc2pktlth / m_packets2.size();
2101                      for (std::deque<Ptr<Packet> >::iterator c =
                         m_packets3.begin(); c != m_packets3.end(); c++) {
2102                  sc3pktlth += (*c)->GetSize();
2103                     }
2104                  sc3pktlth = sc3pktlth / m_packets3.size();
2105                  SC1BUCKETS =0;
2106                  SC2BUCKETS =((i2/gcd) *
                     Case1_conf::DestinationLINKCAPACITY)/(100*8*sc2pktlth);
2107                  SC3BUCKETS =((i3/gcd) *
                     Case1_conf::DestinationLINKCAPACITY)/(100*8*sc3pktlth);
2108                  std::cout<<"stat 4"<<" "<<linkID<<"
                     "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                     "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                     "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                     "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
2109
2110             }
2111         }    //i1==1
2112     } else {
2113         if (i2 == 0) {
```

```
2114                if (i3 == 0) {
2115                    // 1,0,0    //5
2116                    gcd = i1;
2117                    for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a
                        != m_packets1.end(); a++) {
2118                     sc1pktlth += (*a)->GetSize();
2119                        }
2120                     sc1pktlth = sc1pktlth / m_packets1.size();
2121                     sc2pktlth=0;
2122                     sc3pktlth=0;
2123                     SC1BUCKETS =(Case1_conf::DestinationLINKCAPACITY)/(8*sc1pktlth);
2124                     SC2BUCKETS =0;
2125                     SC3BUCKETS =0;
2126                     std::cout<<"stat 5"<<" "<<linkID<<"
                         "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                         "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                         "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                         "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
2127
2128
2129                } else {
2130                    // 1,0,1     //6
2131                    gcd = calculateGCD(i1, i3);
2132                    for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a
                        != m_packets1.end(); a++) {
2133                     sc1pktlth += (*a)->GetSize();
2134                        }
2135                     sc1pktlth = sc1pktlth / m_packets1.size();
2136                     sc2pktlth=0;
2137                     for (std::deque<Ptr<Packet> >::iterator c = m_packets3.begin(); c
                        != m_packets3.end(); c++) {
2138                     sc3pktlth += (*c)->GetSize();
2139                        }
2140                     sc3pktlth = sc3pktlth / m_packets3.size();
2141                     SC1BUCKETS =((i1/gcd) *
                         Case1_conf::DestinationLINKCAPACITY)/(100*8*sc1pktlth);
2142                     SC2BUCKETS =0;
2143                     SC3BUCKETS =((i3/gcd) *
                         Case1_conf::DestinationLINKCAPACITY)/(100*8*sc3pktlth);
2144                     std::cout<<"stat 6"<<" "<<linkID<<"
                         "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"
                         "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"
                         "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"
                         "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
2145
2146                }
2147            } else {
2148                if (i3 == 0) {
2149                    // 1,1,0    //7
2150                    gcd = calculateGCD(i1, i2);
2151                    for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a
                        != m_packets1.end(); a++) {
2152                     sc1pktlth += (*a)->GetSize();
2153                        }
2154                     sc1pktlth = sc1pktlth / m_packets1.size();
2155                         for (std::deque<Ptr<Packet> >::iterator b =
                             m_packets2.begin(); b != m_packets2.end(); b++) {
2156                     sc2pktlth += (*b)->GetSize();
2157                        }
2158                     sc2pktlth = sc2pktlth / m_packets2.size();
2159                     sc3pktlth =0;
```

```
2160                SC1BUCKETS =((i1/gcd) *                                          ↵
                    Case1_conf::DestinationLINKCAPACITY)/(100*8*sc1pktlth);
2161                SC2BUCKETS =((i2/gcd) *                                          ↵
                    Case1_conf::DestinationLINKCAPACITY)/(100*8*sc2pktlth);
2162                SC3BUCKETS =0;
2163               std::cout<<"stat 7"<<" "<<linkID<<"                               ↵
                   "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"              ↵
                   "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"                 ↵
                   "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"      ↵
                   "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
2164
2165
2166            } else {
2167                // 1,1,1   //8
2168                gcd = calculateGCD(calculateGCD(i1, i2), i3);
2169                for (std::deque<Ptr<Packet> >::iterator a = m_packets1.begin(); a  ↵
                    != m_packets1.end(); a++) {
2170                 sc1pktlth += (*a)->GetSize();
2171                    }
2172                 sc1pktlth = sc1pktlth / m_packets1.size();
2173                     for (std::deque<Ptr<Packet> >::iterator b =                  ↵
                         m_packets2.begin(); b != m_packets2.end(); b++) {
2174                 sc2pktlth += (*b)->GetSize();
2175                    }
2176                 sc2pktlth = sc2pktlth / m_packets2.size();
2177                     for (std::deque<Ptr<Packet> >::iterator c =                  ↵
                         m_packets3.begin(); c != m_packets3.end(); c++) {
2178                 sc3pktlth += (*c)->GetSize();
2179                    }
2180                 sc3pktlth = sc3pktlth / m_packets3.size();
2181                 SC1BUCKETS =((i1/gcd) *                                          ↵
                    Case1_conf::DestinationLINKCAPACITY)/(100*8*sc1pktlth);
2182                 SC2BUCKETS =((i2/gcd) *                                          ↵
                    Case1_conf::DestinationLINKCAPACITY)/(100*8*sc2pktlth);
2183                 SC3BUCKETS =((i3/gcd) *                                          ↵
                    Case1_conf::DestinationLINKCAPACITY)/(100*8*sc3pktlth);
2184                std::cout<<"stat 8"<<" "<<linkID<<"                               ↵
                   "<<Simulator::Now()/1000000000<<"  "<<"sc1pktlth"<<"              ↵
                   "<<sc1pktlth<<" "<<"sc2pktlth"<<" "<<sc2pktlth<<"                 ↵
                   "<<"sc3pktlth"<<" "<<sc3pktlth<<" "<<"i1"<<"  "<<i1<<"i2"<<"      ↵
                   "<<i2<<"  "<<"i3"<<" " <<i3<<std::endl;
2185
2186            }
2187        }
2188    }
2189    NS_LOG_INFO("NEW SCHEDULE RATE BUCKETS " << linkID << " " <<"                 ↵
    "<<Case1_conf::DestinationLINKCAPACITY<<"  "<< gcd << " " << SC1BUCKETS << " "  ↵
    << SC2BUCKETS << " "
2190                                            << SC3BUCKETS);
2191
2192
2193 }
2194 // For fixed weights links
2195 else if (linkID == 145 ||  linkID == 189 || // Scenairo 1 Fixed weights queues
2196           linkID == 245 ||  linkID == 2811 || linkID == 2911 || linkID == 21011 ||  ↵
           // Scenairo 2 Fixed weights queues
2197           linkID == 334 ||  linkID == 367 || linkID == 31013 || linkID == 31113 ||  ↵
           // Scenairo 3 Fixed weights queues
2198            linkID == 31213)
2199 {
2200            gcd =1;
```

```
2201            SC1BUCKETS =5;
2202            SC2BUCKETS =4;
2203            SC3BUCKETS =3;
2204            NS_LOG_INFO("NEW SCHEDULE RATE BUCKETS " << linkID << " " <<"
                "<<Case1_conf::LINKCAPACITY<<"  "<< gcd << " " << SC1BUCKETS << " " <<
                SC2BUCKETS << " "
2205                                            << SC3BUCKETS);
2206
2207    }
2208
2209
2210        SC1BUCKETSCPY = SC1BUCKETS;
2211        SC2BUCKETSCPY = SC2BUCKETS;
2212        SC3BUCKETSCPY = SC3BUCKETS;
2213
2214       NS_LOG_FUNCTION(linkID<<"  "<<"average pkt size"<<" "<<sc1pktlth<<"
       "<<sc2pktlth<<" "<<sc3pktlth);
2215       NS_LOG_FUNCTION(linkID<<"  "<<"i1:"<<"
       "<<i1<<"i2:"<<i2<<"i3:"<<i3<<"GCD"<<gcd<<Case1_conf::LINKCAPACITY);
2216
2217
2218        return true;
2219    }
2220
2221    // Set Scheduling Rate By Factor Function. It is used to determine the congestion
       level in service class queue only without managing resources between Diffserv
       domains
2222    bool RDWQueue::SetSchedulingRateByFactor(int band, float alpha)
2223    {
2224        if (band == 1) {
2225            SetSchedulingRateByFactorSC1 = true;
2226            if (alpha <= SC1CONGESTIONFATORALPHAL) {
2227                NS_LOG_INFO("ByFactor case 1"
2228                            << " " << linkID << " " << band << " " << alpha);
2229                NS_LOG_INFO("SC1 case 1");
2230            } else if (alpha > SC1CONGESTIONFATORALPHAL && alpha <=
               SC1CONGESTIONFATORALPHAH) {
2231                NS_LOG_INFO("ByFactor case 2"
2232                            << " " << linkID << " " << band << " " << alpha);
2233
2234                if (alpha <= (SC1CONGESTIONFATORALPHAL + SC1CONGESTIONFATORALPHAH) / 2) {
2235                    NS_LOG_INFO("SC1 case 2");
2236                } else {
2237                    NS_LOG_INFO("SC1 case 3");
2238                }
2239            } else if (alpha > SC1CONGESTIONFATORALPHAH) {
2240                NS_LOG_INFO("ByFactor case 3"
2241                            << " " << linkID << " " << band << " " << alpha);
2242                NS_LOG_INFO("SC1 case 4");
2243            }
2244        } else if (band == 2) {
2245            SetSchedulingRateByFactorSC2 = true;
2246            if (alpha <= SC2CONGESTIONFATORALPHAL) {
2247                NS_LOG_INFO("ByFactor case 1"
2248                            << " " << linkID << " " << band << " " << alpha);
2249                NS_LOG_INFO("SC2 case 1");
2250            } else if (alpha > SC2CONGESTIONFATORALPHAL && alpha <=
               SC2CONGESTIONFATORALPHAH) {
2251                NS_LOG_INFO("ByFactor case 2"
2252                            << " " << linkID << " " << band << " " << alpha);
2253                if (alpha <= (SC2CONGESTIONFATORALPHAL + SC2CONGESTIONFATORALPHAH) / 2) {
```

```
2254                    NS_LOG_INFO("SC2 case 2");
2255                } else {
2256                    NS_LOG_INFO("SC2 case 3");
2257                }
2258            } else if (alpha > SC2CONGESTIONFATORALPHAH) {
2259                NS_LOG_INFO("ByFactor case 3"
2260                            << " " << linkID << " " << band << " " << alpha);
2261                NS_LOG_INFO("SC2 case 4");
2262            }
2263        } else if (band == 3) {
2264            SetSchedulingRateByFactorSC3 = true;
2265            if (alpha <= SC3CONGESTIONFATORALPHAL) {
2266                NS_LOG_INFO("ByFactor case 1"
2267                            << " " << linkID << " " << band << " " << alpha);
2268                NS_LOG_INFO("SC3 case 1");
2269            } else if (alpha > SC3CONGESTIONFATORALPHAL && alpha <=
    SC3CONGESTIONFATORALPHAH) {
2270                NS_LOG_INFO("ByFactor case 2"
2271                            << " " << linkID << " " << band << " " << alpha);
2272                if (alpha <= (SC3CONGESTIONFATORALPHAL + SC3CONGESTIONFATORALPHAH) / 2) {
2273                    NS_LOG_INFO("SC3 case 2");
2274                } else {
2275                    NS_LOG_INFO("SC3 case 3");
2276                }
2277            } else if (alpha > SC3CONGESTIONFATORALPHAH) {
2278                NS_LOG_INFO("ByFactor case 3"
2279                            << " " << linkID << " " << band << " " << alpha);
2280                NS_LOG_INFO("SC3 case 4");
2281            }
2282        }
2283        if (SetSchedulingRateByFactorSC1 && SetSchedulingRateByFactorSC2 &&
    SetSchedulingRateByFactorSC3) {
2284            SetSchedulingRateByFactorSC1 = SetSchedulingRateByFactorSC2 =
    SetSchedulingRateByFactorSC3 = false;
2285            CalculateBucket(SC1ScheRate, SC2ScheRate, SC3ScheRate);
2286        }
2287        return true;
2288    }
2289
2290    // Show Statistic Report Function. It is used to display statistical information
    related to service class queues within an interval k.
2291    void RDWQueue::ShowStatisticReport(void)
2292    {
2293        // report the status
2294        // avg queue delay
2295
2296        float sc1avgdelay = 0;
2297
2298        for (std::list<uint64_t>::iterator it = SC1Delay.begin(); it !=
    SC1Delay.end(); it++) {
2299            sc1avgdelay += *it;
2300        }
2301        sc1avgdelay = sc1avgdelay / SC1Delay.size();
2302        if (std::isnan(sc1avgdelay))
2303        {
2304            sc1avgdelay=0;
2305        }
2306
2307        SC1Delay.clear();
2308
2309
```

```
2310
2311        float sc2avgdelay = 0;
2312        for (std::list<uint64_t>::iterator it = SC2Delay.begin(); it !=
            SC2Delay.end(); it++) {
2313            sc2avgdelay += *it;
2314        }
2315        sc2avgdelay = sc2avgdelay / SC2Delay.size();
2316        if (std::isnan(sc2avgdelay))
2317        {
2318            sc2avgdelay=0;
2319        }
2320        SC2Delay.clear();
2321
2322
2323
2324        float sc3avgdelay = 0;
2325        for (std::list<uint64_t>::iterator it = SC3Delay.begin(); it !=
            SC3Delay.end(); it++) {
2326            sc3avgdelay += *it;
2327        }
2328        sc3avgdelay = sc3avgdelay / SC3Delay.size();
2329        if (std::isnan(sc3avgdelay))
2330        {
2331            sc3avgdelay=0;
2332        }
2333      SC3Delay.clear();
2334
2335
2336
2337        // avg queue length
2338        // scheduling rate
2339        float sc1avgquelen = 0;
2340        for (std::list<double>::iterator it = SC1Usage.begin(); it != SC1Usage.end();
            it++) {
2341             sc1avgquelen = *it;  // to display the instantonous current length in
                 byte for whole period of simulation .
2342
2343        }
2344
2345
2346        float sc2avgquelen = 0;
2347        for (std::list<double>::iterator it = SC2Usage.begin(); it != SC2Usage.end();
            it++) {
2348            sc2avgquelen = *it;
2349
2350        }
2351
2352        float sc3avgquelen = 0;
2353        for (std::list<double>::iterator it = SC3Usage.begin(); it != SC3Usage.end();
            it++) {
2354            sc3avgquelen = *it;
2355
2356        }
2357
2358        NS_LOG_INFO("QueueReport " << linkID << "  " << sc1avgdelay << "  " <<
            sc2avgdelay << "  " << sc3avgdelay << "  "
2359                                   << sc1avgquelen << "  " << sc2avgquelen << "  " <<
                                       sc3avgquelen << "  " << sc1ByteLength
2360                                   << "  " << sc2ByteLength << "  " << sc3ByteLength);
2361    }
2362    // Set buffer size for service class queue in bytes
```

```
2363  void RDWQueue::setBYTEBUFFER(uint32_t sc1, uint32_t sc2, uint32_t sc3)
2364  {
2365      SC1BYTEBUFFER = sc1;
2366      SC2BYTEBUFFER = sc2;
2367      SC3BYTEBUFFER = sc3;
2368  }
2369
2370  // Set the higher threshold parameter of the WRED for service class queue.
2371
2372  void RDWQueue::setREDTHIGH(double sc1, double sc2, double sc3)
2373  {
2374      REDSC1THIGH = sc1;
2375      REDSC2THIGH = sc2;
2376      REDSC3THIGH = sc3;
2377  }
2378  // Set Higher drop probability parameter of the WRED for each service class queue.
2379  void RDWQueue::setREDHDROP(double sc1, double sc2, double sc3)
2380  {
2381      REDSC1HDROP = sc1;
2382      REDSC2HDROP = sc2;
2383      REDSC3HDROP = sc3;
2384  }
2385  // Set the lower threshold parameter of the WRED for service class queue.
2386
2387  void RDWQueue::setREDTLOW(double sc1, double sc2, double sc3)
2388  {
2389      REDSC1TLOW = sc1;
2390      REDSC2TLOW = sc2;
2391      REDSC3TLOW = sc3;
2392  }
2393
2394  // Set Lower drop probability parameter of the WRED for each service class queue.
2395
2396  void RDWQueue::setREDLDROP(double sc1, double sc2, double sc3)
2397  {
2398      REDSC1LDROP = sc1;
2399      REDSC2LDROP = sc2;
2400      REDSC3LDROP = sc3;
2401  }
2402  // Set the lower threshold parameter for the congestion level in each service
     class.
2403  void RDWQueue::setALPHAL(float sc1, float sc2, float sc3)
2404  {
2405      SC1CONGESTIONFATORALPHAL = sc1;
2406      SC2CONGESTIONFATORALPHAL = sc2;
2407      SC3CONGESTIONFATORALPHAL = sc3;
2408  }
2409
2410  // Set the higher threshold parameter for the congestion level in each service
     class.
2411
2412  void RDWQueue::setALPHAH(float sc1, float sc2, float sc3)
2413  {
2414      SC1CONGESTIONFATORALPHAH = sc1;
2415      SC2CONGESTIONFATORALPHAH = sc2;
2416      SC3CONGESTIONFATORALPHAH = sc3;
2417  }
2418  // Set fixed beta level (Not considered in the research)
2419  void RDWQueue::setBELTA(float sc1, float sc2, float sc3)
2420  {
2421      CONGESTIONFATBELTA1 = sc1;
```

```
2422        CONGESTIONFATBELTA2 = sc2;
2423        CONGESTIONFATBELTA3 = sc3;
2424    }
2425    // Calculating the scheduling rates for service class queues at the router with multi out ports
2426    bool RDWQueue::SetSchedulingRateByAverage(int band, double sc1averagelength, double sc2averagelength, double sc3averagelength, double sc1averagedelay,double sc2averagedelay, double sc3averagedelay )
2427
2428    {
2429        double SC1ScheRateAvg, SC2ScheRateAvg,  SC3ScheRateAvg;
2430
2431        if (sc1averagedelay!=0 && sc1averagelength!=0 && sc2averagedelay!=0 && sc2averagelength!=0 && sc3averagedelay!=0 && sc3averagelength!=0) //1
2432        {
2433         SC1ScheRateAvg = sc1averagelength /(sc1averagedelay/1000) ;
2434         SC2ScheRateAvg = 0.5 * SC1ScheRateAvg * sc2averagelength / sc1averagelength;
2435         SC3ScheRateAvg =  0.25 *SC1ScheRateAvg * sc3averagelength / sc1averagelength;
2436        }
2437        else if (sc1averagedelay==0 || sc1averagelength==0) //2
2438        {
2439         SC1ScheRateAvg=0;
2440         SC2ScheRateAvg = sc2averagelength / (sc2averagedelay/1000);
2441         SC3ScheRateAvg = 0.5 * SC2ScheRateAvg * sc3averagelength / sc2averagelength;
2442        }
2443        else if (sc2averagedelay==0 || sc2averagelength==0)  //3
2444        {
2445         SC1ScheRateAvg = sc1averagelength / (sc1averagedelay/1000);
2446         SC2ScheRateAvg=0;
2447         SC3ScheRateAvg = 0.5 * SC1ScheRateAvg * sc3averagelength / sc1averagelength;
2448        }
2449         else if (sc3averagedelay==0 || sc3averagelength==0) //4
2450        {
2451         SC1ScheRateAvg = sc1averagelength / (sc1averagedelay/1000);
2452         SC2ScheRateAvg = 0.5 * SC1ScheRateAvg * sc2averagelength / sc1averagelength;
2453         SC3ScheRateAvg=0;
2454        }
2455        else if ((sc1averagedelay==0 || sc1averagelength==0) && (sc2averagedelay==0 || sc2averagelength==0) )  //5
2456        {
2457         SC1ScheRateAvg=0;
2458         SC2ScheRateAvg=0;
2459         SC3ScheRateAvg = sc3averagelength/(sc3averagedelay/1000);
2460        }
2461        else if ((sc1averagedelay==0 || sc1averagelength==0) && (sc3averagedelay==0 || sc3averagelength==0))  //6
2462        {
2463         SC1ScheRateAvg=0;
2464         SC2ScheRateAvg = sc2averagelength / (sc2averagedelay/1000);
2465         SC3ScheRateAvg=0;
2466        }
2467        else if ((sc2averagedelay==0 || sc2averagelength==0) && (sc3averagedelay==0 || sc3averagelength==0) )  //7
2468        {
2469         SC1ScheRateAvg = sc1averagelength / (sc1averagedelay/1000);
2470         SC2ScheRateAvg=0;
2471         SC3ScheRateAvg=0;
2472
2473        }
2474
2475        else if (sc1averagedelay==0 && sc1averagelength==0 && sc2averagedelay==0 &&
```

```
    sc2averagelength==0 && sc3averagedelay==0 && sc3averagelength==0)  //8
    {
        SC1ScheRateAvg = SC2ScheRateAvg = SC3ScheRateAvg =0;
    }

if (std::isnan(SC1ScheRateAvg) || std::isinf(SC1ScheRateAvg))
        {
        SC1ScheRateAvg=0;
        }
        else if (std::isnan(SC2ScheRateAvg) || std::isinf(SC2ScheRateAvg))
         {
        SC2ScheRateAvg=0;
         }
         else if (std::isnan(SC3ScheRateAvg) || std::isinf(SC3ScheRateAvg))
         {
        SC3ScheRateAvg=0;
         }


    if (band == 1) {
        SetSchedulingRateByAverageSC1 = true;
        SC1ScheRate = SC1ScheRateAvg ;
        NS_LOG_INFO("SC1 rate is set to Average");
     } else if (band == 2) {
        SetSchedulingRateByAverageSC2 = true;
        SC2ScheRate = SC2ScheRateAvg;
        NS_LOG_INFO("SC2 rate is set to Average");
       } else if (band == 3) {
        SetSchedulingRateByAverageSC3 = true;
        SC3ScheRate = SC3ScheRateAvg;
        NS_LOG_INFO("SC3 rate is set to Average");
       }


  if (SetSchedulingRateByAverageSC1 && SetSchedulingRateByAverageSC2 &&
  SetSchedulingRateByAverageSC3) {
        SetSchedulingRateByAverageSC1 = SetSchedulingRateByAverageSC2 =
        SetSchedulingRateByAverageSC3 = false;
        CalculateBucket(SC1ScheRate, SC2ScheRate, SC3ScheRate);
        NS_LOG_INFO("CalculateSchedulingRateByAverage " << linkID << " " <<
        SC1ScheRate << " " << SC2ScheRate << " " << SC3ScheRate);
        NS_LOG_INFO("Average Length Values " << linkID << " " << band << " " <<
        sc1averagelength << " " << sc2averagelength << " " <<sc3averagelength);
        NS_LOG_INFO("Average Delay Values " << linkID << " " << band << " " <<
        sc1averagedelay <<" " << sc2averagedelay << " " << sc3averagedelay);

    }
    return true;

}
// To get the average queue delay for each service class queue at the router with
multiple out ports
void RDWQueue::UpdateAverageQueueDelay(int band, double AverageQueueDelay)
{
    if (band ==1)
    {
    if (AverageQueueDelay>=0){
    SC1CurrAverageDelay.push_back(AverageQueueDelay);}
    else SC1CurrAverageDelay.push_back(0);
    NS_LOG_INFO("SC1UpdateAverageQueueDelay" << linkID << " " <<
    SC1CurrAverageDelay.back());
```

```
2529      }
2530
2531      else if (band ==2)
2532      {
2533      if (AverageQueueDelay>=0){
2534      SC2CurrAverageDelay.push_back(AverageQueueDelay);}
2535      else SC2CurrAverageDelay.push_back(0);
2536      NS_LOG_INFO("SC2UpdateAverageQueueDelay" << linkID << " " <<
          SC2CurrAverageDelay.back());
2537      }
2538
2539      else if (band ==3)
2540      {
2541      if (AverageQueueDelay>=0){
2542      SC3CurrAverageDelay.push_back(AverageQueueDelay);}
2543      else    SC3CurrAverageDelay.push_back(0);
2544      NS_LOG_INFO("SC3UpdateAverageQueueDelay" << linkID << " " <<
          SC3CurrAverageDelay.back());
2545      }
2546  }
2547  // Used in the process of selecting minumum Average queue delay for each service
      class queue at the router with multiple out ports to calculate the service class
      scheduling rate at the router with multiple out ports.
2548  double RDWQueue::GetAverageQueueDelay(int band)
2549  {   double AverageQueueDelay;
2550      if (band == 1)
2551      {
2552          AverageQueueDelay= SC1CurrAverageDelay.front();
2553          NS_LOG_INFO("SC1GetAverageQueueDelay" << linkID << " " << AverageQueueDelay);
2554          SC1CurrAverageDelay.erase (SC1CurrAverageDelay.begin());
2555          if (AverageQueueDelay>=0)
2556          {
2557          return AverageQueueDelay;
2558          }
2559      }
2560      else if (band ==2)
2561      {
2562          AverageQueueDelay= SC2CurrAverageDelay.front();
2563          NS_LOG_INFO("SC2GetAverageQueueDelay" << linkID << " " << AverageQueueDelay);
2564          SC2CurrAverageDelay.erase (SC2CurrAverageDelay.begin());
2565          if (AverageQueueDelay>=0)
2566          {
2567          return AverageQueueDelay;
2568          }
2569      }
2570          else if (band ==3)
2571      {
2572          AverageQueueDelay= SC3CurrAverageDelay.front();
2573          NS_LOG_INFO("SC3GetAverageQueueDelay" << linkID << " " << AverageQueueDelay);
2574          SC3CurrAverageDelay.erase (SC3CurrAverageDelay.begin());
2575          if (AverageQueueDelay>=0)
2576          {
2577          return AverageQueueDelay;
2578          }
2579      }
2580      return 0;
2581
2582  }
2583  // To get the average queue length for each service class queue at the router with
      multiple out ports (used in the process of managing resources between DiffServ
      domains to calculate the maximum congestion level in service classs queue)
```

```
2584
2585  void RDWQueue::UpdateAverageQueueLength1 (int band, double AverageQueueLength)
2586  {
2587      if (band ==1)
2588      {
2589      SC1CurrAverageLength1.push_back(AverageQueueLength);
2590      NS_LOG_INFO("SC1UpdateAverageQueueLength" << linkID << " " <<
          SC1CurrAverageLength1.back());
2591      }
2592      else if (band ==2)
2593      {
2594      SC2CurrAverageLength1.push_back(AverageQueueLength);
2595      NS_LOG_INFO("SC2UpdateAverageQueueLength" << linkID << " " <<
          SC2CurrAverageLength1.back());
2596      }
2597      else if (band ==3)
2598      {
2599      SC3CurrAverageLength1.push_back(AverageQueueLength);
2600      NS_LOG_INFO("SC3UpdateAverageQueueLength" << linkID << " " <<
          SC3CurrAverageLength1.back());
2601      }
2602  }
2603  // Used in the process of selecting maximum Average queue length for each service
      class queue at the router with multiple out ports to specify the maximum
      congestion level.
2604
2605  double RDWQueue::GetAverageQueueLength1(int band)
2606  {   double AverageQueueLength1;
2607      if (band == 1)
2608      {
2609          AverageQueueLength1= SC1CurrAverageLength1.front();
2610          NS_LOG_INFO("SC1GetAverageQueueLength" << linkID << " " <<
              AverageQueueLength1);
2611          SC1CurrAverageLength1.erase (SC1CurrAverageLength1.begin());
2612          return AverageQueueLength1;
2613
2614      }
2615      else if (band ==2)
2616      {
2617          AverageQueueLength1= SC2CurrAverageLength1.front();
2618          NS_LOG_INFO("SC2GetAverageQueueLength" << linkID << " " <<
              AverageQueueLength1);
2619          SC2CurrAverageLength1.erase (SC2CurrAverageLength1.begin());
2620          return AverageQueueLength1;
2621
2622      }
2623          else if (band ==3)
2624      {
2625          AverageQueueLength1= SC3CurrAverageLength1.front();
2626          NS_LOG_INFO("SC3GetAverageQueueLength" << linkID << " " <<
              AverageQueueLength1);
2627          SC3CurrAverageLength1.erase (SC3CurrAverageLength1.begin());
2628          return AverageQueueLength1;
2629
2630      }
2631      return 0;
2632
2633  }
2634  // To get the average queue length for each service class queue at the router with
      multiple out ports
2635
```

```
2636  void RDWQueue::UpdateAverageQueueLength2 (int band, double AverageQueueLength)
2637  {
2638      if (band ==1)
2639      {
2640      SC1CurrAverageLength2.push_back(AverageQueueLength);
2641      NS_LOG_INFO("SC1UpdateAverageQueueLength" << linkID << " " <<
      SC1CurrAverageLength2.back());
2642      }
2643      else if (band ==2)
2644      {
2645      SC2CurrAverageLength2.push_back(AverageQueueLength);
2646      NS_LOG_INFO("SC2UpdateAverageQueueLength" << linkID << " " <<
      SC2CurrAverageLength2.back());
2647      }
2648      else if (band ==3)
2649      {
2650      SC3CurrAverageLength2.push_back(AverageQueueLength);
2651      NS_LOG_INFO("SC3UpdateAverageQueueLength" << linkID << " " <<
      SC3CurrAverageLength2.back());
2652      }
2653  }
2654  // Used in the process of selecting maximum Average queue length for each service
      class queue at the router with multiple out ports to calculate the service class
      scheduling rate at the router with multiple out ports .
2655
2656  double RDWQueue::GetAverageQueueLength2(int band)
2657  {   double AverageQueueLength2;
2658      if (band == 1)
2659      {
2660          AverageQueueLength2= SC1CurrAverageLength2.front();
2661          NS_LOG_INFO("SC1GetAverageQueueLength" << linkID << " " <<
          AverageQueueLength2);
2662          SC1CurrAverageLength2.erase (SC1CurrAverageLength2.begin());
2663          return AverageQueueLength2;
2664
2665      }
2666      else if (band ==2)
2667      {
2668          AverageQueueLength2= SC2CurrAverageLength2.front();
2669          NS_LOG_INFO("SC2GetAverageQueueLength" << linkID << " " <<
          AverageQueueLength2);
2670          SC2CurrAverageLength2.erase (SC2CurrAverageLength2.begin());
2671          return AverageQueueLength2;
2672
2673      }
2674          else if (band ==3)
2675      {
2676          AverageQueueLength2= SC3CurrAverageLength2.front();
2677          NS_LOG_INFO("SC3GetAverageQueueLength" << linkID << " " <<
          AverageQueueLength2);
2678          SC3CurrAverageLength2.erase (SC3CurrAverageLength2.begin());
2679          return AverageQueueLength2;
2680
2681      }
2682      return 0;
2683
2684  }
2685  // Managing the service classes scheduling rates between DiffServ domains function
      based on the congestion level and scheduling rate update factor (beta)
2686  bool RDWQueue::SetSchedulingRateByLinearFactor(int band, float avgqueuelength)
2687  {
```

```
2688        if (band == 1) {
2689            SetSchedulingRateByLinearFactorSC1 = true;
2690            float sc1alpha = avgqueuelength/SC1BYTEBUFFER;
2691            if (sc1alpha <= SC1CONGESTIONFATORALPHAL) {
2692                NS_LOG_INFO("ByLinearFactor case 1"
2693                            << " " << linkID << " " << band <<"  "<< "alpha value:"
                                <<"  "<< sc1alpha <<"  "<<"beta value:" <<" " << 0);
2694                SC1ScheRate = SC1ScheRate;
2695                NS_LOG_INFO("SC1 case 1");
2696            } else if (sc1alpha > SC1CONGESTIONFATORALPHAL && sc1alpha <
                SC1CONGESTIONFATORALPHAH) {
2697                float sc1beta =
                    Case1_conf::SC1BETAMAX*((sc1alpha-SC1CONGESTIONFATORALPHAL)/(SC1CONGESTI
                    ONFATORALPHAH-SC1CONGESTIONFATORALPHAL));
2698
2699                NS_LOG_INFO("ByLinearFactor case 2"
2700                            << " " << linkID << " " << band <<"  "<< "alpha value:"
                                <<"  "<< sc1alpha <<"  "<< "Beta value:" <<"  "<< sc1beta);
2701                SC1ScheRate = SC1ScheRate - sc1beta * SC1ScheRate;
2702
2703             }
2704             else { NS_LOG_INFO("ByLinearFactor case 3"
2705                            << " " << linkID << " " << band <<"  "<< "alpha value:"
                                <<"  "<< sc1alpha <<"  "<< "beta value:" <<"  "<<
                                Case1_conf::SC1BETAMAX);
2706                 SC1ScheRate = SC1ScheRate - (SC1ScheRate * Case1_conf::SC1BETAMAX);
2707                    NS_LOG_INFO("SC1 case 3");
2708             }
2709         }
2710
2711
2712        if (band == 2) {
2713            SetSchedulingRateByLinearFactorSC2 = true;
2714            float sc2alpha = avgqueuelength/SC2BYTEBUFFER;
2715            if (sc2alpha <= SC2CONGESTIONFATORALPHAL) {
2716                NS_LOG_INFO("ByLinearFactor case 1"
2717                            << " " << linkID << " " << band <<"  "<< "alpha value:"
                                <<"  "<< sc2alpha <<"  "<<"beta value:" <<"  "<< 0);
2718                SC2ScheRate = SC2ScheRate;
2719                NS_LOG_INFO("SC2 case 1");
2720            } else if (sc2alpha > SC2CONGESTIONFATORALPHAL && sc2alpha <
                SC2CONGESTIONFATORALPHAH) {
2721                float sc2beta = Case1_conf::SC2BETAMAX *
                    (sc2alpha-SC2CONGESTIONFATORALPHAL)/(SC2CONGESTIONFATORALPHAH-SC2CONGEST
                    IONFATORALPHAL);
2722                NS_LOG_INFO("ByLinearFactor case 2"
2723                            << " " << linkID << " " << band <<"  "<< "alpha value:"
                                <<"  "<< sc2alpha <<"  "<< "Beta value:"<<"  " << sc2beta);
2724                SC2ScheRate = SC2ScheRate - sc2beta * SC2ScheRate;
2725             }
2726             else { NS_LOG_INFO("ByLinearFactor case 3"
2727                            << " " << linkID << " " << band <<"  "<< "alpha
                                value:"<<"  " << sc2alpha <<"  "<< "beta value:" <<"
                                "<<Case1_conf::SC2BETAMAX);
2728                 SC2ScheRate = SC2ScheRate - (SC2ScheRate * Case1_conf::SC2BETAMAX);
2729                    NS_LOG_INFO("SC2 case 3");
2730             }
2731         }
2732
2733             if (band == 3) {
2734            SetSchedulingRateByLinearFactorSC3 = true;
```

```
2735          float sc3alpha = avgqueuelength/SC3BYTEBUFFER;
2736          if (sc3alpha <= SC3CONGESTIONFATORALPHAL) {
2737              NS_LOG_INFO("ByLinearFactor case 1"
2738                          << " " << linkID << " " << band <<"  "<< "alpha value:"
                            <<"  "<< sc3alpha <<"  "<<"beta value:" <<"  "<< 0);
2739              SC3ScheRate = SC3ScheRate;
2740              NS_LOG_INFO("SC3 case 1");
2741          } else if (sc3alpha > SC3CONGESTIONFATORALPHAL && sc3alpha <
              SC3CONGESTIONFATORALPHAH) {
2742              float sc3beta = Case1_conf::SC3BETAMAX *
                  (sc3alpha-SC3CONGESTIONFATORALPHAL)/(SC3CONGESTIONFATORALPHAH-SC3CONGEST
                  IONFATORALPHAL);
2743              NS_LOG_INFO("ByLinearFactor case 2"
2744                          << " " << linkID << " " << band <<"  "<< "alpha value:"
                            <<"  "<< sc3alpha <<"  "<< "Beta value:" <<"  "<< sc3beta);
2745              SC3ScheRate = SC3ScheRate - sc3beta * SC3ScheRate;
2746           }
2747           else { NS_LOG_INFO("ByLinearFactor case 3"
2748                          << " " << linkID << " " << band <<"  "<< "alpha value:"
                            <<"  "<< sc3alpha <<"  "<< "beta value:" <<"  "<<
                            Case1_conf::SC3BETAMAX);
2749                SC3ScheRate = SC3ScheRate - (SC3ScheRate * Case1_conf::SC3BETAMAX);
2750                   NS_LOG_INFO("SC3 case 3");
2751           }
2752       }
2753
2754      if (SetSchedulingRateByLinearFactorSC1 && SetSchedulingRateByLinearFactorSC2 &&
      SetSchedulingRateByLinearFactorSC3) {
2755           SetSchedulingRateByLinearFactorSC1 = SetSchedulingRateByLinearFactorSC2 =
          SetSchedulingRateByLinearFactorSC3 = false;
2756           CalculateBucket(SC1ScheRate, SC2ScheRate, SC3ScheRate);
2757       }
2758       return true;
2759   }
2760
```

A-4: Traffic Model Test NS3 Simulation Code:

```
#ifndef TOSAPP_H
#define TOSAPP_H

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/csma-module.h"
#include "ns3/qos-tag.h"


using namespace ns3;

class TosApp : public Application
{
public:
    //TosApp();
    TosApp(uint8_t tos, uint32_t TrafPatt);
    virtual ~TosApp();


    void Setup (Ptr<Socket> socket, Address address, uint32_t packetSize, uint32_t
    nPackets, DataRate dataRate);
    void Setup(Ptr<Socket> socket, Address address, uint32_t packetSize, uint32_t
    nPackets, DataRate dataRate, double mean, double bound, int onofftype);

private:
    virtual void StartApplication (void);
    virtual void StopApplication (void);

    void ScheduleTx (void);
    void SendPacket (void);
    uint32_t m_onofftype; // 1=random value 2= fixed value
    double m_ontime;
    double m_offtime;
    double m_fliptimestamp;
    double interval;
    uint32_t m_currentONOFFstat; // 1=on  0=off

    Ptr<Socket>     m_socket;
    Address         m_peer;
    uint32_t        m_packetSize;
    uint32_t        m_nPackets;
    DataRate        m_dataRate;
    EventId         m_sendEvent;
    bool            m_running;
    uint32_t        m_packetsSent;
    QosTag          qostag;
    uint32_t        trafficPattern;
};

#endif // TOSAPP_H

```

```cpp
#include "TosApp.h"
#include "ns3/random-variable-stream.h"
#include "Case1_conf.h"
#include "ns3/ipv4-header.h"

using namespace ns3;



TosApp::TosApp(uint8_t tos, uint32_t TrafPatt)
    : m_socket(0)
    , m_peer()
    , m_packetSize(0)
    , m_nPackets(0)
    , m_dataRate(0)
    , m_sendEvent()
    , m_running(false)
    , m_packetsSent(0)
    , trafficPattern(TrafPatt)
{
    qostag.SetTid(tos);
}

TosApp::~TosApp()
{
    m_socket = 0;
}

void TosApp::Setup(Ptr<Socket> socket, Address address, uint32_t packetSize,
uint32_t nPackets, DataRate dataRate)
{
    m_socket = socket;
    m_peer = address;
    m_packetSize = packetSize;       //packet size
    m_nPackets = nPackets;
    //m_nPackets = (rand() / double(RAND_MAX)) * nPackets;
    m_dataRate = dataRate;
    // edit 24/08/2016
    m_onofftype = 3;
    m_currentONOFFstat = 1; // on stat

    if (m_onofftype == 1) { // varies
        Ptr<ExponentialRandomVariable> x = CreateObject<ExponentialRandomVariable>();
        x->SetAttribute("Mean", DoubleValue(Case1_conf::ONTIMEEXPONMEAN));
        x->SetAttribute("Bound", DoubleValue(Case1_conf::ONTIMEEXPONBOUND));
        m_fliptimestamp = Simulator::Now().GetSeconds() + x->GetValue();

    } else if (m_onofftype == 2) { // fixed
        Ptr<ExponentialRandomVariable> x = CreateObject<ExponentialRandomVariable>();
        x->SetAttribute("Mean", DoubleValue(Case1_conf::ONTIMEEXPONMEAN));
        x->SetAttribute("Bound", DoubleValue(Case1_conf::ONTIMEEXPONBOUND));
        m_ontime = x->GetValue();
        m_fliptimestamp = Simulator::Now().GetSeconds() + m_ontime;

        x = CreateObject<ExponentialRandomVariable>();
        x->SetAttribute("Mean", DoubleValue(Case1_conf::OFFTIMEEXPONMEAN));
        x->SetAttribute("Bound", DoubleValue(Case1_conf::OFFTIMEEXPONBOUND));
        m_offtime = x->GetValue();
        std::cout << m_ontime << " " << m_offtime << std::endl;
    } else {
        m_fliptimestamp = 1000000000;
```

```
61      }//////////////////////
62  }
63  // traffic with on-off period
64  void TosApp::Setup(Ptr<Socket> socket,
65                     Address address,
66                     uint32_t packetSize,
67                     uint32_t nPackets,
68                     DataRate dataRate,
69                     double mean,
70                     double bound,
71                     int onofftype)
72  {
73      m_socket = socket;
74      m_peer = address;
75      m_packetSize = packetSize;
76      m_nPackets = (rand() / double(RAND_MAX)) * nPackets;
77      m_dataRate = dataRate;
78
79      if (trafficPattern == 0) {
80          // constant
81          Case1_conf::CONSTANTRADVAR = mean;
82      } else if (trafficPattern == 1) {
83          // exponential
84          Case1_conf::EXPONRADVARMEAN = mean;
85          Case1_conf::EXPONRADVARBOUND = bound;
86      } else if (trafficPattern == 2) {
87          // pareto
88          Case1_conf::PARETORADVARMEAN = mean;
89          Case1_conf::PARETORADVARSAHPE = bound;
90      }
91      m_onofftype = onofftype;
92      m_currentONOFFstat = 1; // on stat
93
94      if (m_onofftype == 1) { // varies
95          Ptr<ExponentialRandomVariable> x = CreateObject<ExponentialRandomVariable>();
96          x->SetAttribute("Mean", DoubleValue(Case1_conf::ONTIMEEXPONMEAN));
97          x->SetAttribute("Bound", DoubleValue(Case1_conf::ONTIMEEXPONBOUND));
98          m_fliptimestamp = Simulator::Now().GetSeconds() + x->GetValue();
99
100     } else if (m_onofftype == 2) { // fixed
101         Ptr<ExponentialRandomVariable> x = CreateObject<ExponentialRandomVariable>();
102         x->SetAttribute("Mean", DoubleValue(Case1_conf::ONTIMEEXPONMEAN));
103         x->SetAttribute("Bound", DoubleValue(Case1_conf::ONTIMEEXPONBOUND));
104         m_ontime = x->GetValue();
105         m_fliptimestamp = Simulator::Now().GetSeconds() + m_ontime;
106
107         x = CreateObject<ExponentialRandomVariable>();
108         x->SetAttribute("Mean", DoubleValue(Case1_conf::OFFTIMEEXPONMEAN));
109         x->SetAttribute("Bound", DoubleValue(Case1_conf::OFFTIMEEXPONBOUND));
110         m_offtime = x->GetValue();
111         std::cout <<"m_ontime"<< m_ontime << " " << "m_offtime"<< m_offtime <<
            std::endl;
112     } else {
113         m_fliptimestamp = 1000000000;
114     }
115 }
116 //////////////
117
118 void TosApp::StartApplication(void)
119 {
120     m_running = true;
```

```
121     m_packetsSent = 0;
122     m_socket->Bind();
123     m_socket->Connect(m_peer);
124     SendPacket();
125 }
126
127 void TosApp::StopApplication(void)
128 {
129     m_running = false;
130
131     if (m_sendEvent.IsRunning()) {
132         Simulator::Cancel(m_sendEvent);
133     }
134
135     if (m_socket) {
136         m_socket->Close();
137     }
138 }
139
140 void TosApp::SendPacket(void)
141 {   // 13/05 edit
142      if (trafficPattern==3)   // VoIP traffic packet size
143     {
144         m_packetSize = 200;
145     }
146     else if (trafficPattern==4)    // Video conferencing traffic packet size
147     {
148     Ptr<ExponentialRandomVariable> x = CreateObject<ExponentialRandomVariable>();
149     x->SetAttribute("Mean", DoubleValue(34.285)); //previous 200
150     x->SetAttribute("Bound", DoubleValue(200));
151     m_packetSize = (uint32_t)(x->GetValue()) + 1300;
152     }
153     else if (trafficPattern==5)    // FTP traffic
154     {
155     Ptr<ExponentialRandomVariable> x = CreateObject<ExponentialRandomVariable>();
156     x->SetAttribute("Mean", DoubleValue(91.4285));  //357
157     x->SetAttribute("Bound", DoubleValue(300)); // amount of difference between
        max. and min.
158     m_packetSize = (uint32_t)(x->GetValue())+200;   // Min. value.
159     }
160     else if (trafficPattern==6)    // Database Query message traffic,Best effort
161     {
162         m_packetSize = 512;
163     }
164
165
166     // random packet size
167
168     /*Ptr<ExponentialRandomVariable> x = CreateObject<ExponentialRandomVariable>();
169     x->SetAttribute("Mean", DoubleValue(Case1_conf::EXPONPKTSIZEMEAN));
170     x->SetAttribute("Bound", DoubleValue(Case1_conf::EXPONPKTSIZEBOUND));
171     m_packetSize = (uint32_t)(x->GetValue() * Case1_conf::TRAFFICPKTSIZE) + 1;*/
172     /*std::cout<< m_socket->GetNode()-> GetId() <<" pkt size "<<m_packetSize<<"
        tos " << (uint32_t)qostag.GetTid ()
173     <<std::endl;*/
174     Ptr<Packet> packet = Create<Packet>(m_packetSize);
175     packet->AddPacketTag(qostag);
176     std::cout << "Pkt_create " << packet->GetUid() <<" "<<Simulator::Now()<<"
        "<<(uint32_t)qostag.GetTid()<< std::endl;
177     m_socket->Send(packet);
178
```

```
179         // For testing the generated traffic
180         std::cout <<"Node ID."<< m_socket->GetNode()-> GetId()<<"  "<<"Pkt_create"<<"
            "<<"pkt size:"<<m_packetSize<<"  "<<packet->GetUid()<<"
            "<<Simulator::Now()<<"  "<<(uint32_t)qostag.GetTid()<< std::endl;
181
182
183         if (++m_packetsSent < m_nPackets) {
184             ScheduleTx();
185         }
186     }
187
188     void TosApp::ScheduleTx(void)
189     {
190         // update of on-off appllication
191          if (Simulator::Now().GetSeconds() >= m_fliptimestamp) {
192             if (m_currentONOFFstat == 1) { // on -> off
193                 if (m_onofftype == 1) {
194                     Ptr<ExponentialRandomVariable> x =
                        CreateObject<ExponentialRandomVariable>();
195                     x->SetAttribute("Mean", DoubleValue(Case1_conf::OFFTIMEEXPONMEAN));
196                     x->SetAttribute("Bound", DoubleValue(Case1_conf::OFFTIMEEXPONBOUND));
197                     double interval = x->GetValue();
198                     while(interval<=0){
199                         interval = x->GetValue();
200                     }
201                     m_fliptimestamp = Simulator::Now().GetSeconds() + interval;
202                 } else if (m_onofftype == 2) {
203                     m_fliptimestamp = Simulator::Now().GetSeconds() + m_offtime;
204                 }
205                 m_currentONOFFstat = 0; // go off
206                 std::cout << "now off " << Simulator::Now().GetSeconds() << " " <<
                    m_fliptimestamp << std::endl;
207             } else { // off -> on
208                 if (m_onofftype == 1) {
209                     Ptr<ExponentialRandomVariable> x =
                        CreateObject<ExponentialRandomVariable>();
210                     x->SetAttribute("Mean", DoubleValue(Case1_conf::ONTIMEEXPONMEAN));
211                     x->SetAttribute("Bound", DoubleValue(Case1_conf::ONTIMEEXPONBOUND));
212                     double interval = x-> GetValue();
213                     while(interval<=0){
214                         interval = x->GetValue();
215                     }
216                     m_fliptimestamp = Simulator::Now().GetSeconds() + interval;
217                 } else if (m_onofftype == 2) {
218                     m_fliptimestamp = Simulator::Now().GetSeconds() + m_ontime;
219                 }
220                 m_currentONOFFstat = 1; // go on
221                 std::cout << "now on " << Simulator::Now().GetSeconds() << " " <<
                    m_fliptimestamp << std::endl;
222             }
223         }
224
225         if (m_currentONOFFstat == 1) // if on, send packets
226         {
227     ////////////////////////////
228         if (m_running) {
229             double value = 0;
230
231             if (trafficPattern == 0) {
232                 // constant
233                 Ptr<ConstantRandomVariable> x = CreateObject<ConstantRandomVariable>();
```

```
234             x->SetAttribute("Constant", DoubleValue(Case1_conf::CONSTANTRADVAR));
235             value = x->GetValue();
236         } else if (trafficPattern == 1) {
237             // exponential
238             Ptr<ExponentialRandomVariable> x =                                    ↵
                CreateObject<ExponentialRandomVariable>();
239             x->SetAttribute("Mean", DoubleValue(Case1_conf::EXPONRADVARMEAN));
240             x->SetAttribute("Bound", DoubleValue(Case1_conf::EXPONRADVARBOUND));
241             value = x->GetValue();
242         } else if (trafficPattern == 2) {
243             // pareto
244             Ptr<ParetoRandomVariable> x = CreateObject<ParetoRandomVariable>();
245             x->SetAttribute("Mean", DoubleValue(Case1_conf::PARETORADVARMEAN));
246             x->SetAttribute("Shape", DoubleValue(Case1_conf::PARETORADVARSAHPE));
247             value = x->GetValue();
248         }
249         //edit 13/05
250         else if (trafficPattern == 3) {
251             // VoIP packets
252             Ptr<ConstantRandomVariable> x = CreateObject<ConstantRandomVariable>();
253             x->SetAttribute("Constant", DoubleValue(0.02));
254             value = x->GetValue();
255             value =value - 0.02;
256             /*Ptr<ExponentialRandomVariable> x =                                  ↵
                CreateObject<ExponentialRandomVariable>();
257             x->SetAttribute("Mean", DoubleValue(0.02));
258             x->SetAttribute("Bound", DoubleValue(0));
259             value = x->GetValue();*/
260         }  else if (trafficPattern == 4) {
261             // Video conferencing packets
262             Ptr<ExponentialRandomVariable> x =                                    ↵
                CreateObject<ExponentialRandomVariable>();
263             x->SetAttribute("Mean", DoubleValue(0.02916)); //0.005
264             x->SetAttribute("Bound", DoubleValue(0.006));
265             value = x->GetValue();
266         }  else if (trafficPattern == 5) {
267             //  traffic (FTP)
268             Ptr<ExponentialRandomVariable> x =                                    ↵
                CreateObject<ExponentialRandomVariable>();
269             x->SetAttribute("Mean", DoubleValue(0.01093));  //0.0028
270             x->SetAttribute("Bound", DoubleValue(0.006));
271             value = x->GetValue();
272         }  else if (trafficPattern == 6) {
273             // Database Query message traffic, Best effort
274             Ptr<ExponentialRandomVariable> x =                                    ↵
                CreateObject<ExponentialRandomVariable>();
275             x->SetAttribute("Mean", DoubleValue(0.05));
276             x->SetAttribute("Bound", DoubleValue(0.006));
277             value = x->GetValue();
278         }
279
280          ////////////
281         Time tNext(Seconds(m_packetSize * 8 /                                     ↵
        static_cast<double>(m_dataRate.GetBitRate()) + value));
282         m_sendEvent = Simulator::Schedule(tNext, &TosApp::SendPacket, this);
283     }
284 }
285 // update of on-off application
286 else {
287         Time tNext(Seconds(m_fliptimestamp-Simulator::Now().GetSeconds()+0.001));
288         Simulator::Schedule(tNext, &TosApp::ScheduleTx, this);
```

```
289         }
290     }
291     ////////////////////////////
292
```

A-5: Test Scenarios Network Topology Simulation using NS3:

The topologies of the test scenarios presented in section 6.3 are built by the execution of the network topology application. In this application, objects from the queuing model application (RDWQueue), the traffic model application (TosApp) and the packet sink application (PacketSink) are created to represent the queues, traffic sources and the destinations of the DiffServ upstream and downstream domains respectively. Any computer network is constructed from multiple elements that perform various functions. These elements can be switches, routers, servers, access points, etc. To represent these elements using the network simulator NS3, this simulation must create a node for each element. Each node has at least one network device that is used to connect it to another node via a link. Any two nodes are connected via a point to point channel or link – if it is a wired network. The point to point channel should also be built and have its attributes defined in terms of delay, rate and inter-frame gap. The type of the network device is based on the type of the channel that is used to connect it to the other node. The NS3 software provides a number of (Helpers) that can be used to instantiate NS3 models (class references) on a node. The (Helpers) that are used in this simulation are as follows.

1. InternetStackHelper this is used to aggregate the IP/TCP/UDP functionalities of existing nodes. In this simulation program, "stack" is an instance of NS3::InternetStackHelper.
2. IpV4AddressHelper – this is a simple IPv4 address generator. It is used to assign simple IPv4 address to node interfaces. In this simulation program, "address" is an instance of NS3::Ipv4AddressHelper.
3. PointToPointHelper – this is used to build a point to point network device object. In this simulation program, "p2pXY" is an instance of NS3::PointToPointHelper which is used to setup a point to point interface link that connects the point to point network devices at nodes X and Y.
4. PacketSinkHelper – this is used to instantiate a packet sink application on the destination node. In this simulation program, "packetSinkHelper" is an instance of NS3::PacketSinkHelperHelper .

Any two nodes (or more) in NS3 represent a network and both will have IP and mask addresses. The IP addresses for the network hosts (nodes) and the addresses of their network devices must also be defined when building a network topology. Within the node, the functionality of the network element is represented. If the element represents a DiffServ router then an object from

RDWQueue is created, provided with an identification, and its attributes will be configured and will be setup inside the node. If the element represents a traffic source or destination then an object from either TosApp or PacketSink is created, provided with an identification, and its attributes will be configured and will be setup inside the node. Before adding a traffic source to a node, the TCP and UDP socket addresses for the source must be created in order to organise the process of sending packets from the traffic source applications to the sink applications at the destination. A "socket" in the context of communication means creating an endpoint for communication. The socket address is a combination of an IP address for the source node and a port number for the destination node. Based on the socket address, the Internet socket delivers incoming data packets to the appropriate application at the destination for processing. The management process of any service class queue is accomplished within the network topology application through using the type of the service class and the identification of the queue. This means that the process of managing resources across different DiffServ domains and the process of configuring the service class scheduling rates for the queues that belong to the respective service class at the ingress routers of the downstream domains – as per scenarios 2 and 3 – are undertaken by this application. These two processes also have to be identified and scheduled by the NS3 scheduler (Simulator::Schedule Class Reference) during the simulation run.

Moreover, the time that a packet is received at the destination is also calculated in the network topology application by comparing the destination address that is set in the packet header with the address of a particular destination node and taking the current simulation time. This time will be used to measure the End to End Delay for each service class' traffic.

A-5.1: Test Scenario 1, Topology Simulation Diagram and NS3 Code:

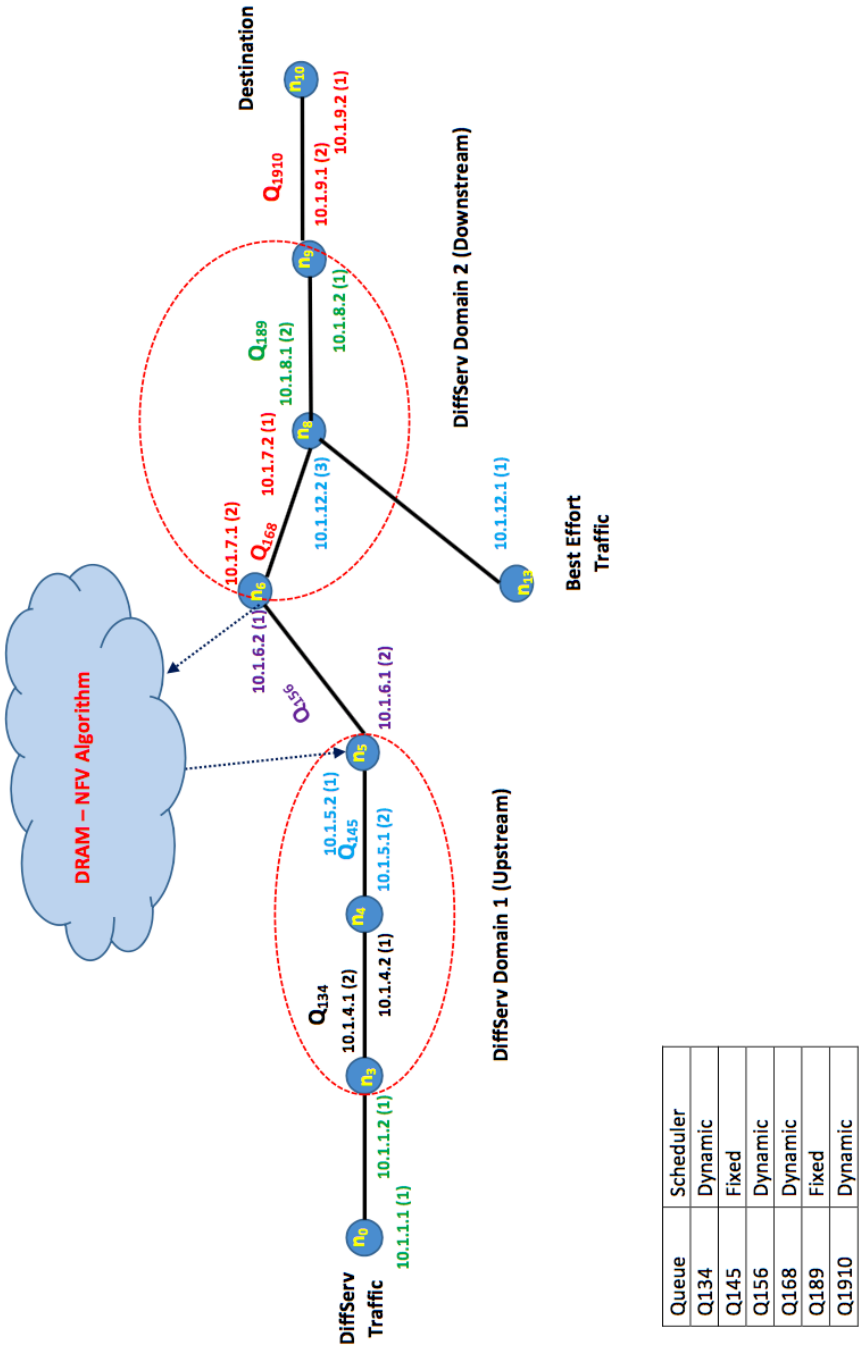


Figure 9-8, test scenario 1 simulation network topology diagram.

```
#ifndef CASE1_H
#define CASE1_H

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/csma-module.h"
#include "RDWQueue.h"
#include "ns3/qos-tag.h"  // added for onoff appl.
using namespace ns3;

class Case1
{
    public:
        Case1();
        virtual ~Case1();

    NodeContainer nodes;
   Ptr<RDWQueue> queue134;
   Ptr<RDWQueue> queue145;
   Ptr<RDWQueue> queue156;
   Ptr<RDWQueue> queue168;
   Ptr<RDWQueue> queue189;
   Ptr<RDWQueue> queue1910;

   void ScheduleSchedulingRateUpdate(void);
   //void TrafficFromTo(int from, int to, int ProtocolType ,int TrafficType, int ToS);
   void TrafficFromTo1(int from, int to, int ProtocolType ,int TrafficType, int ToS);
   void TrafficFromTo2(int from, int to, int ProtocolType ,int TrafficType, int ToS);
   void TrafficFromTo3(int from, int to, int ProtocolType ,int TrafficType, int ToS);

   void OnOffTrafficFromTo(int from, int to, int ProtocolType, int TrafficType, int ToS); // added for
onoff app.
   QosTag qosTag; //added for onoff appl.


};

#endif // CASE1_H
```

```cpp
#include "Case1.h"
#include "TosApp.h"
#include "Case1_conf.h"


using namespace ns3;

// static void MacRecv(Ptr<const Packet> packet)
static void MacRecv(const Ptr<const Packet> packet, Ptr<Ipv4> ipv4, const uint32_t interface)
{
    Ipv4Header header;
    packet->PeekHeader(header);
    if (header.GetDestination().IsEqual(ipv4->GetAddress(interface, 0).GetLocal())) {
        QosTag tosTag;
        uint32_t band = 0;
        if (packet->PeekPacketTag(tosTag)) {
            band = (uint32_t)tosTag.GetTid();
        }
        if (band != 0 ) {
            std::cout << "Pkt_recv " << packet->GetUid() << " " << Simulator::Now() << " " << band << std::endl;
        }
    }
}

Case1::Case1()
{
    nodes.Create(14);

    InternetStackHelper stack;
    stack.Install(nodes);

    Ipv4AddressHelper address;
    address.SetBase("10.1.1.0", "255.255.255.252");

    // 0-3
    PointToPointHelper p2p03;
    p2p03.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
    p2p03.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
    address.Assign(p2p03.Install(nodes.Get(0), nodes.Get(3)));
    // 1-3
    address.SetBase("10.1.2.0", "255.255.255.252");
    PointToPointHelper p2p13;
    p2p13.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
    p2p13.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
    address.Assign(p2p13.Install(nodes.Get(1), nodes.Get(3)));
    // 2-3
    address.SetBase("10.1.3.0", "255.255.255.252");
    PointToPointHelper p2p23;
    p2p23.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
    p2p23.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
    address.Assign(p2p23.Install(nodes.Get(2), nodes.Get(3)));

    // 3-4
    // notice: queue need to be set before install!!!
    // you can't get the queue from outside the queue, so you need to update the parameter in the queue function.
    // you need to set InterframeGap to decide the queue speed

    address.SetBase("10.1.4.0", "255.255.255.252");
```

```
59      PointToPointHelper p2p34;
60      p2p34.SetChannelAttribute("Delay",                                          ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
61      p2p34.SetDeviceAttribute("DataRate",                                        ↵
        DataRateValue(DataRate(Case1_conf::UpstreamLINKCAPACITY)));
62      p2p34.SetDeviceAttribute("InterframeGap", TimeValue(MilliSeconds(0)));
63      queue134 = CreateObject<RDWQueue>(134);
64      queue134->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,  ↵
        Case1_conf::SC3BYTEBUFFER);
65      queue134->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,       ↵
        Case1_conf::REDSC3THIGH);
66      queue134->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,       ↵
        Case1_conf::REDSC3HDROP);
67      queue134->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,          ↵
        Case1_conf::REDSC3TLOW);
68      queue134->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,       ↵
        Case1_conf::REDSC3LDROP);
69      queue134->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
70                          Case1_conf::SC2CONGESTIONFATORALPHAL,
71                          Case1_conf::SC3CONGESTIONFATORALPHAL);
72      queue134->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
73                          Case1_conf::SC2CONGESTIONFATORALPHAH,
74                          Case1_conf::SC3CONGESTIONFATORALPHAH);
75      queue134->setBELTA(
76          Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,        ↵
            Case1_conf::CONGESTIONFATBELTA3);
77      address.Assign(p2p34.Install(nodes.Get(3), nodes.Get(4)));
78                                                                                  ↵
        nodes.Get(3)->GetDevice(4)->GetObject<PointToPointNetDevice>()->SetQueue(queue13↵
        4);
79      // 4-5
80
81      address.SetBase("10.1.5.0", "255.255.255.252");
82      PointToPointHelper p2p45;
83      p2p45.SetChannelAttribute("Delay",                                          ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
84      p2p45.SetDeviceAttribute("DataRate",                                        ↵
        DataRateValue(DataRate(Case1_conf::UpstreamLINKCAPACITY)));
85      p2p45.SetDeviceAttribute("InterframeGap", TimeValue(MilliSeconds(0.5)));
86      queue145 = CreateObject<RDWQueue>(145);
87      queue145->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,  ↵
        Case1_conf::SC3BYTEBUFFER);
88      queue145->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,       ↵
        Case1_conf::REDSC3THIGH);
89      queue145->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,       ↵
        Case1_conf::REDSC3HDROP);
90      queue145->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,          ↵
        Case1_conf::REDSC3TLOW);
91      queue145->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,       ↵
        Case1_conf::REDSC3LDROP);
92      queue145->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
93                          Case1_conf::SC2CONGESTIONFATORALPHAL,
94                          Case1_conf::SC3CONGESTIONFATORALPHAL);
95      queue145->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
96                          Case1_conf::SC2CONGESTIONFATORALPHAH,
97                          Case1_conf::SC3CONGESTIONFATORALPHAH);
98      queue145->setBELTA(
99          Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,        ↵
            Case1_conf::CONGESTIONFATBELTA3);
100     address.Assign(p2p45.Install(nodes.Get(4), nodes.Get(5)));
101                                                                                 ↵
```

```
    nodes.Get(4)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue145);
102
103     // 5-6
104
105     address.SetBase("10.1.6.0", "255.255.255.252");
106     PointToPointHelper p2p56;
107     p2p56.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
108     p2p56.SetDeviceAttribute("DataRate", DataRateValue(DataRate(Case1_conf::DomainsLINKCAPACITY)));
109     p2p56.SetDeviceAttribute("InterframeGap", TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP)));
110     queue156 = CreateObject<RDWQueue>(156);
111     queue156->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER, Case1_conf::SC3BYTEBUFFER);
112     queue156->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH, Case1_conf::REDSC3THIGH);
113     queue156->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP, Case1_conf::REDSC3HDROP);
114     queue156->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW, Case1_conf::REDSC3TLOW);
115     queue156->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP, Case1_conf::REDSC3LDROP);
116     queue156->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
117                         Case1_conf::SC2CONGESTIONFATORALPHAL,
118                         Case1_conf::SC3CONGESTIONFATORALPHAL);
119     queue156->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
120                         Case1_conf::SC2CONGESTIONFATORALPHAH,
121                         Case1_conf::SC3CONGESTIONFATORALPHAH);
122     queue156->setBELTA(
123         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2, Case1_conf::CONGESTIONFATBELTA3);
124     address.Assign(p2p56.Install(nodes.Get(5), nodes.Get(6)));
125
    nodes.Get(5)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue156);
126
127     // 6-8
128
129     address.SetBase("10.1.7.0", "255.255.255.252");
130     PointToPointHelper p2p68;
131     p2p68.SetChannelAttribute("Delay", TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
132     p2p68.SetDeviceAttribute("DataRate", DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
133     p2p68.SetDeviceAttribute("InterframeGap", TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+2)));
134     queue168 = CreateObject<RDWQueue>(168);
135     queue168->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER, Case1_conf::SC3BYTEBUFFER);
136     queue168->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH, Case1_conf::REDSC3THIGH);
137     queue168->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP, Case1_conf::REDSC3HDROP);
138     queue168->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW, Case1_conf::REDSC3TLOW);
139     queue168->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP, Case1_conf::REDSC3LDROP);
140     queue168->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
141                         Case1_conf::SC2CONGESTIONFATORALPHAL,
142                         Case1_conf::SC3CONGESTIONFATORALPHAL);
```

```
143     queue168->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
144                         Case1_conf::SC2CONGESTIONFATORALPHAH,
145                         Case1_conf::SC3CONGESTIONFATORALPHAH);
146     queue168->setBELTA(
147         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);
148     address.Assign(p2p68.Install(nodes.Get(6), nodes.Get(8)));
149
        nodes.Get(6)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue168);
150
151     // 8-9
152
153     address.SetBase("10.1.8.0", "255.255.255.252");
154     PointToPointHelper p2p89;
155     p2p89.SetChannelAttribute("Delay",
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
156     p2p89.SetDeviceAttribute("DataRate",
        DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
157     p2p89.SetDeviceAttribute("InterframeGap",
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+4)));
158     queue189 = CreateObject<RDWQueue>(189);
159     queue189->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
        Case1_conf::SC3BYTEBUFFER);
160     queue189->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
        Case1_conf::REDSC3THIGH);
161     queue189->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
        Case1_conf::REDSC3HDROP);
162     queue189->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
        Case1_conf::REDSC3TLOW);
163     queue189->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
        Case1_conf::REDSC3LDROP);
164     queue189->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
165                         Case1_conf::SC2CONGESTIONFATORALPHAL,
166                         Case1_conf::SC3CONGESTIONFATORALPHAL);
167     queue189->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
168                         Case1_conf::SC2CONGESTIONFATORALPHAH,
169                         Case1_conf::SC3CONGESTIONFATORALPHAH);
170     queue189->setBELTA(
171         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);
172     address.Assign(p2p89.Install(nodes.Get(8), nodes.Get(9)));
173
        nodes.Get(8)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue189);
174     // 9-10
175
176     address.SetBase("10.1.9.0", "255.255.255.252");
177     PointToPointHelper p2p910;
178     p2p910.SetChannelAttribute("Delay",
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
179     p2p910.SetDeviceAttribute("DataRate",
        DataRateValue(DataRate(Case1_conf::DestinationLINKCAPACITY)));
180      p2p910.SetDeviceAttribute("InterframeGap",
         TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+6)));
181     queue1910 = CreateObject<RDWQueue>(1910);
182     queue1910->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
        Case1_conf::SC3BYTEBUFFER);
183     queue1910->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
        Case1_conf::REDSC3THIGH);
184     queue1910->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
```

```
    Case1_conf::REDSC3HDROP);
185 queue1910->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
    Case1_conf::REDSC3TLOW);
186 queue1910->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
    Case1_conf::REDSC3LDROP);
187 queue1910->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
188                      Case1_conf::SC2CONGESTIONFATORALPHAL,
189                      Case1_conf::SC3CONGESTIONFATORALPHAL);
190 queue1910->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
191                      Case1_conf::SC2CONGESTIONFATORALPHAH,
192                      Case1_conf::SC3CONGESTIONFATORALPHAH);
193 queue1910->setBELTA(
194     Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
    Case1_conf::CONGESTIONFATBELTA3);
195 address.Assign(p2p910.Install(nodes.Get(9), nodes.Get(10)));
196
    nodes.Get(9)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue19
    10);
197
198
199 // 9-11
200
201 address.SetBase("10.1.10.0", "255.255.255.252");
202 PointToPointHelper p2p911;
203 p2p911.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
204 p2p911.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000)));
205 address.Assign(p2p911.Install(nodes.Get(9), nodes.Get(11)));
206
207 // 9-12
208
209 address.SetBase("10.1.11.0", "255.255.255.252");
210 PointToPointHelper p2p912;
211 p2p912.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
212 p2p912.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000)));
213 address.Assign(p2p912.Install(nodes.Get(9), nodes.Get(12)));
214
215    // 13-8
216 address.SetBase("10.1.12.0", "255.255.255.252");
217 PointToPointHelper p2p138;
218 p2p138.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
219 p2p138.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
220 address.Assign(p2p138.Install(nodes.Get(13), nodes.Get(8)));
221
222 for (uint32_t i = 0; i < nodes.GetN(); i++) {
223     for (uint32_t j = 1; j < nodes.Get(i)->GetNDevices(); j++) {
224     std::cout<<i<<" "<<nodes.Get(i)->GetObject<Ipv4> ()->GetAddress (j,
    0).GetLocal ()<<"Device"<<nodes.Get(i)->GetNDevices()<<"Get device
    index"<<nodes.Get(i) -> GetDevice(j)<<std::endl;
225
226     }
227 }
228
229 // sink
230 uint16_t sinkPort = 8080;
231
232 PacketSinkHelper udpSinkHelper("ns3::UdpSocketFactory",
    InetSocketAddress(Ipv4Address::GetAny(), sinkPort));
233 PacketSinkHelper tcpSinkHelper10tcp("ns3::TcpSocketFactory",
    InetSocketAddress(Ipv4Address::GetAny(), sinkPort));
234
235 ApplicationContainer sinkApps10 = udpSinkHelper.Install(nodes.Get(10));
```

```
236      sinkApps10.Start(Seconds(0.));
237      ApplicationContainer sinkApps10tcp = tcpSinkHelper10tcp.Install(nodes.Get(10));
238      sinkApps10tcp.Start(Seconds(0.));
239
240      ApplicationContainer sinkApps11 = udpSinkHelper.Install(nodes.Get(11));
241      sinkApps11.Start(Seconds(0.));
242      ApplicationContainer sinkApps11tcp = tcpSinkHelper10tcp.Install(nodes.Get(11));
243      sinkApps11tcp.Start(Seconds(0.));
244
245      ApplicationContainer sinkApps12 = udpSinkHelper.Install(nodes.Get(12));
246      sinkApps12.Start(Seconds(0.));
247      ApplicationContainer sinkApps12tcp = tcpSinkHelper10tcp.Install(nodes.Get(12));
248      sinkApps12tcp.Start(Seconds(0.));
249
250
251  // traffic 0
252       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                    ↵
         1);TrafficFromTo3(0, 10, 3, 3, 1);
253       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                    ↵
         1);TrafficFromTo3(0, 10, 3, 3, 1);
254       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                    ↵
         1);TrafficFromTo3(0, 10, 3, 3, 1);
255       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                    ↵
         1);TrafficFromTo3(0, 10, 3, 3, 1);
256       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                    ↵
         1);TrafficFromTo3(0, 10, 3, 3, 1);
257       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                    ↵
         1);TrafficFromTo3(0, 10, 3, 3, 1);
258        TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
259        TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
260        TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
261        TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
262
263      TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                    ↵
         1);TrafficFromTo3(0, 10, 3, 3, 1);
264       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
265       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
266       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
267       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
268       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
269       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
270       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
271       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
272       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
          1);TrafficFromTo3(0, 10, 3, 3, 1);
273
274      TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                    ↵
         1);TrafficFromTo3(0, 10, 3, 3, 1);
275       TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                   ↵
```

```
      1);TrafficFromTo3(0, 10, 3, 3, 1);
276   TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                ↵
      1);TrafficFromTo3(0, 10, 3, 3, 1);
277   TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                ↵
      1);TrafficFromTo3(0, 10, 3, 3, 1);
278   TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                ↵
      1);TrafficFromTo3(0, 10, 3, 3, 1);
279   TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                ↵
      1);TrafficFromTo3(0, 10, 3, 3, 1);
280   TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                ↵
      1);TrafficFromTo3(0, 10, 3, 3, 1);
281   TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                ↵
      1);TrafficFromTo3(0, 10, 3, 3, 1);
282   TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                ↵
      1);TrafficFromTo3(0, 10, 3, 3, 1);
283   TrafficFromTo3(0, 10, 3, 3, 1);TrafficFromTo3(0, 10, 3, 3,                ↵
      1);TrafficFromTo3(0, 10, 3, 3, 1);
284
285
286
287   //+++++++++++++++++
288     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
289     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
290     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
291     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
292     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
293     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
294     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
295     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
296     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
297     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
298
299     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
300     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
301     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
302     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
303     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
304     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
305     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
306     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
307     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
308     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,              ↵
```

```
        2);TrafficFromTo3(0, 10, 4, 4, 2);
309
310     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,                ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
311     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,                ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
312     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,                ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
313     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,                ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
314     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,                ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
315     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,                ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
316     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,                ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
317     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,                ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
318     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,                ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
319     TrafficFromTo3(0, 10, 4, 4, 2);TrafficFromTo3(0, 10, 4, 4,                ↵
        2);TrafficFromTo3(0, 10, 4, 4, 2);
320
321
322
323     //++++++++++++++++++++
324     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
325     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
326     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
327     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
328     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
329     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
330     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
331     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
332     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
333     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
334
335     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
336     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
337     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
338     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
339     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
340     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
341     TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,                ↵
        3);TrafficFromTo3(0, 10, 5, 5, 3);
```

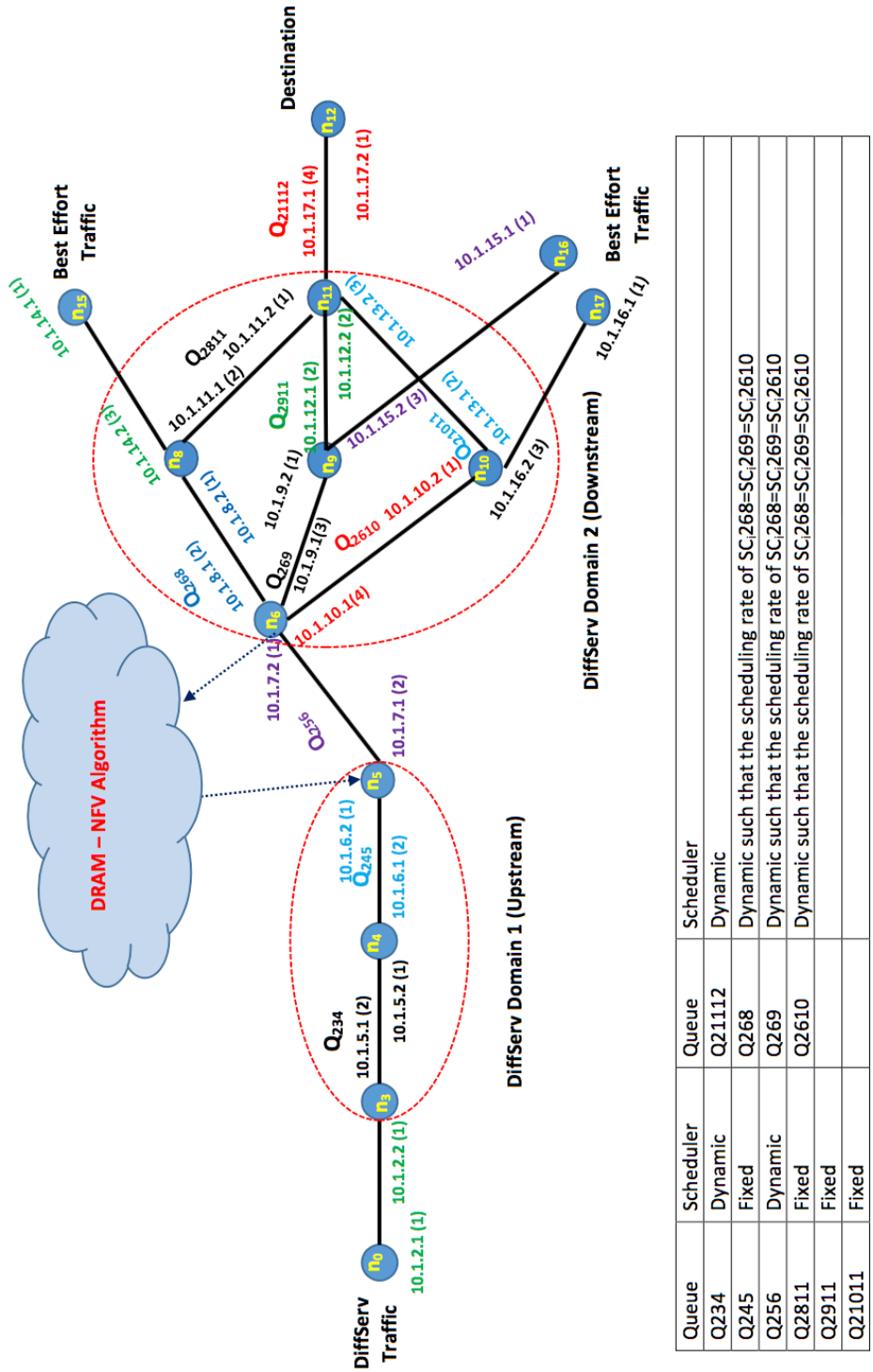
```
342         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
343         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
344         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
345
346         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
347         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
348         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
349         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
350         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
351         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
352         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
353         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
354         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
355         TrafficFromTo3(0, 10, 5, 5, 3);TrafficFromTo3(0, 10, 5, 5,
            3);TrafficFromTo3(0, 10, 5, 5, 3);
356
357
358
359
360     // SC4 traffic
361       TrafficFromTo3(13, 10, 6, 6, 4);
362       TrafficFromTo3(13, 10, 6, 6, 4);
363
364
365
366 //============================================================
367
368     // routing
369     Ipv4GlobalRoutingHelper::PopulateRoutingTables();
370
371     for (double i = 20000; i < Case1_conf::SIM_STOP_Sec * 1000; i += 1000) {
372
373         Simulator::Schedule(MilliSeconds(i), &Case1::ScheduleSchedulingRateUpdate,
            this);
374
375
376     }
377
378     Config::ConnectWithoutContext("/NodeList/*/$ns3::Ipv4L3Protocol/Rx",
    MakeCallback(&MacRecv));
379
380 }
381
382 ////////////// on-off Application - scenairo 1
383 void Case1::TrafficFromTo1(int from, int to, int ProtocolType, int TrafficType,
    int ToS)
384 {
385     uint16_t sinkPort = 8080;
386     Address
```

```
    sinkAddress(InetSocketAddress(nodes.Get(to)->GetObject<Ipv4>()->GetAddress(1,
    0).GetLocal(), sinkPort));
    Ptr<Socket> TrafficSocket;
    Ptr<TosApp> app1 = CreateObject<TosApp>(ToS, TrafficType);
    if (ProtocolType == 3) { // edit 13/05   VoIP traffic
        TrafficSocket = Socket::CreateSocket(nodes.Get(from),
        UdpSocketFactory::GetTypeId());
        app1->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
        (80000/1600)*Case1_conf::SIM_STOP_Sec, DataRate("80000"),1,1,1); //125000

    } else if (ProtocolType == 4) { // edit 13/05   Video Conferencing traffic
        TrafficSocket = Socket::CreateSocket(nodes.Get(from),
        UdpSocketFactory::GetTypeId());
        app1->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
        (384000/11200)*Case1_conf::SIM_STOP_Sec, DataRate("384000"),1,1,1);
        //500000     //previous rate 2240000
        // video conferencing rate 128-384 kbits/sec,
        https://en.wikipedia.org/wiki/Bit_rate
    } else if (ProtocolType == 5) { // edit 13/05    (FTP) traffic  TcpSocketFactory
        TrafficSocket =
        Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
        app1->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
        (256000/2800)*Case1_conf::SIM_STOP_Sec, DataRate("256000"),1,1,1);
        //179000, // previous rate 10000000

    } else if (ProtocolType == 6) { // edit 13/05   // Database Query message
    traffic - Best effort traffic
        TrafficSocket =
        Socket::CreateSocket(nodes.Get(from),UdpSocketFactory::GetTypeId());
        app1->Setup(TrafficSocket, sinkAddress,
        Case1_conf::TRAFFICPKTSIZE,(81920/4096)*Case1_conf::SIM_STOP_Sec ,
        DataRate("81920"),1,1,1); //50000

    }
    nodes.Get(from)->AddApplication(app1);

    app1->SetStartTime(Seconds(1.));
}
void Case1::TrafficFromTo2(int from, int to, int ProtocolType, int TrafficType,
int ToS)
{
    uint16_t sinkPort = 8080;
    Address
    sinkAddress(InetSocketAddress(nodes.Get(to)->GetObject<Ipv4>()->GetAddress(1,
    0).GetLocal(), sinkPort));
    Ptr<Socket> TrafficSocket;
    Ptr<TosApp> app2 = CreateObject<TosApp>(ToS, TrafficType);
    if (ProtocolType == 3) { // edit 13/05   VoIP traffic
        TrafficSocket = Socket::CreateSocket(nodes.Get(from),
        UdpSocketFactory::GetTypeId());
        app2->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
        (80000/1600)*Case1_conf::SIM_STOP_Sec, DataRate("80000"),1,1,2);

    } else if (ProtocolType == 4) { // edit 13/05   Video Conferencing traffic
        TrafficSocket = Socket::CreateSocket(nodes.Get(from),
        UdpSocketFactory::GetTypeId());
        app2->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
        (384000/11200)*Case1_conf::SIM_STOP_Sec, DataRate("384000"),1,1,2);
        //previous rate 2240000
        // video conferencing rate 128-384 kbits/sec,
        https://en.wikipedia.org/wiki/Bit_rate
```

```
424     } else if (ProtocolType == 5) { // edit 13/05    (FTP) traffic  TcpSocketFactory
425         TrafficSocket =                                                          ↵
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
426         app2->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,      ↵
            (256000/2800)*Case1_conf::SIM_STOP_Sec, DataRate("256000"),1,1,2);
427
428     } else if (ProtocolType == 6) { // edit 13/05   // Database Query message    ↵
    traffic - Best effort traffic
429         TrafficSocket =                                                          ↵
            Socket::CreateSocket(nodes.Get(from),UdpSocketFactory::GetTypeId());
430         app2->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,      ↵
            (81920/4096)*Case1_conf::SIM_STOP_Sec, DataRate("81920"),1,1,2);
431     }
432     nodes.Get(from)->AddApplication(app2);
433
434     app2->SetStartTime(Seconds(1.));
435 }
436 void Case1::TrafficFromTo3(int from, int to, int ProtocolType, int TrafficType,  ↵
    int ToS)
437 {
438     uint16_t sinkPort = 8080;
439     Address                                                                      ↵
        sinkAddress(InetSocketAddress(nodes.Get(to)->GetObject<Ipv4>()->GetAddress(1, ↵
        0).GetLocal(), sinkPort));
440     Ptr<Socket> TrafficSocket;
441     Ptr<TosApp> app3 = CreateObject<TosApp>(ToS, TrafficType);
442     if (ProtocolType == 3) { // edit 13/05   VoIP traffic
443         TrafficSocket = Socket::CreateSocket(nodes.Get(from),                    ↵
            UdpSocketFactory::GetTypeId());
444         app3->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,      ↵
            (80000/1600)*Case1_conf::SIM_STOP_Sec, DataRate("80000"),1,1,3); //125000
445
446     } else if (ProtocolType == 4) { // edit 13/05   Video Conferencing traffic
447         TrafficSocket = Socket::CreateSocket(nodes.Get(from),                    ↵
            UdpSocketFactory::GetTypeId());
448         app3->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,      ↵
            (384000/11200)*Case1_conf::SIM_STOP_Sec, DataRate("384000"),1,1,3);      ↵
            //500000  //previous rate 2240000
449         // video conferencing rate 128-384 kbits/sec,                            ↵
            https://en.wikipedia.org/wiki/Bit_rate
450     }
451     else if (ProtocolType == 5) { // edit 13/05    (FTP) traffic  TcpSocketFactory
452         TrafficSocket =                                                          ↵
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
453         app3->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,      ↵
            (256000/2800)*Case1_conf::SIM_STOP_Sec, DataRate("256000"),1,1,3); //179000
454
455     }   else if (ProtocolType == 6) { // edit 13/05   // Database Query message  ↵
    traffic - Best effort traffic
456         TrafficSocket =                                                          ↵
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
457         app3->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,      ↵
            (81920/4096)*Case1_conf::SIM_STOP_Sec, DataRate("81920"),1,1,3); //50000
458
459     }
460     nodes.Get(from)->AddApplication(app3);
461
462     app3->SetStartTime(Seconds(1.));
463
464
465 }
```

```
466 ////////////////////////////
467 void Case1::ScheduleSchedulingRateUpdate(void)
468 {
469    //either section A or section B work during running the simulation.
470    // Section A
471   queue156->SetSchedulingRateByFactor(1, queue168->GetQueueSizeBytes(1) /
    (float)Case1_conf::SC1BYTEBUFFER);
472   queue156->SetSchedulingRateByFactor(2, queue168->GetQueueSizeBytes(2) /
    (float)Case1_conf::SC2BYTEBUFFER);
473   queue156->SetSchedulingRateByFactor(3, queue168->GetQueueSizeBytes(3) /
    (float)Case1_conf::SC3BYTEBUFFER);
474
475    // Section B
476     //set by linear factor
477    queue156->SetSchedulingRateByLinearFactor(1, queue168->GetQueueSizeBytes(1));
478    queue156->SetSchedulingRateByLinearFactor(2, queue168->GetQueueSizeBytes(2));
479    queue156->SetSchedulingRateByLinearFactor(3, queue168->GetQueueSizeBytes(3));
480
481
482
483
484 }
485
486 Case1::~Case1()
487 {
488     // dtor
489 }
490
491
```

A-5.2: Test Scenario 2, Topology Simulation Diagram and NS3 Code:



Queue	Scheduler
Q234	Dynamic
Q245	Fixed
Q256	Dynamic
Q2811	Fixed
Q2911	Fixed
Q21011	Fixed

Queue	Scheduler
Q21112	Dynamic
Q268	Dynamic such that the scheduling rate of $SC_i268=SC_i269=SC_i2610$
Q269	Dynamic such that the scheduling rate of $SC_i268=SC_i269=SC_i2610$
Q2610	Dynamic such that the scheduling rate of $SC_i268=SC_i269=SC_i2610$

Figure 9-9, test scenario 2 simulation network topology diagram.

```cpp
#ifndef CASE2_H
#define CASE2_H

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/csma-module.h"
#include "RDWQueue.h"

using namespace ns3;

class Case2
{
    public:
        Case2();
        virtual ~Case2();

    NodeContainer nodes;

   Ptr<RDWQueue> queue234;
   Ptr<RDWQueue> queue245;
   Ptr<RDWQueue> queue256;
   Ptr<RDWQueue> queue268;
   Ptr<RDWQueue> queue269;
   Ptr<RDWQueue> queue2610;
   Ptr<RDWQueue> queue2811;
   Ptr<RDWQueue> queue2911;
   Ptr<RDWQueue> queue21011;
   Ptr<RDWQueue> queue21112;

    void ScheduleSchedulingRateUpdate(void);
    //void TrafficFromTo(int from, int to, int ProtocolType ,int TrafficType, int ToS);
    void TrafficFromTo1(int from, int to, int ProtocolType ,int TrafficType, int ToS);
    void TrafficFromTo2(int from, int to, int ProtocolType ,int TrafficType, int ToS);
    void TrafficFromTo3(int from, int to, int ProtocolType ,int TrafficType, int ToS);
    void AverageScheduleRate(void);

  double MinimumAverageQueueDelay(double x, double y, double z);
  double MaximumAverageQueueLength(double x, double y, double z);

};

#endif // CASE2_H
```

```cpp
#include "Case2.h"
#include "TosApp.h"
#include "Case1_conf.h"

using namespace ns3;

static void MacRecv(const Ptr<const Packet> packet, Ptr<Ipv4> ipv4, const uint32_t interface)
{
    Ipv4Header header;
    packet->PeekHeader(header);
    if (header.GetDestination().IsEqual(ipv4->GetAddress(interface, 0).GetLocal())) {
        QosTag tosTag;
        uint32_t band = 0;
        if (packet->PeekPacketTag(tosTag)) {
            band = (uint32_t)tosTag.GetTid();
        }
        if (band != 0 ) {
            std::cout << "Pkt_recv " << packet->GetUid() << " " << Simulator::Now() << " " << band << std::endl;
        }
    }
}

Case2::Case2()
{
    Config::SetDefault("ns3::Ipv4GlobalRouting::RandomEcmpRouting", BooleanValue(true));
    nodes.Create(18);

    InternetStackHelper stack;
    stack.Install(nodes);

    Ipv4AddressHelper address;
    address.SetBase("10.1.1.0", "255.255.255.0");

    // 0-3
    address.NewNetwork();
    PointToPointHelper p2p03;
    p2p03.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
    p2p03.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
    address.Assign(p2p03.Install(nodes.Get(0), nodes.Get(3)));



    //1-3  Not Used
    address.NewNetwork();
    PointToPointHelper p2p13;
    p2p13.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
    p2p13.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
    address.Assign(p2p13.Install(nodes.Get(1), nodes.Get(3)));

    //2-3  Not Used
    address.NewNetwork();
    PointToPointHelper p2p23;
    p2p23.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
    p2p23.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
    address.Assign(p2p23.Install(nodes.Get(2), nodes.Get(3)));

    // 3-4
    address.NewNetwork();
```

```
59      PointToPointHelper p2p34;
60      p2p34.SetChannelAttribute("Delay",                                          ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
61      p2p34.SetDeviceAttribute("DataRate",                                        ↵
        DataRateValue(DataRate(Case1_conf::UpstreamLINKCAPACITY)));
62      p2p34.SetDeviceAttribute("InterframeGap", TimeValue(MilliSeconds(0)));
63      address.Assign(p2p34.Install(nodes.Get(3), nodes.Get(4)));
64
65      //4-5
66      address.NewNetwork();
67      PointToPointHelper p2p45;
68      p2p45.SetChannelAttribute("Delay",                                          ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
69      p2p45.SetDeviceAttribute("DataRate",                                        ↵
        DataRateValue(DataRate(Case1_conf::UpstreamLINKCAPACITY)));
70      p2p45.SetDeviceAttribute("InterframeGap", TimeValue(MilliSeconds(0.5)));
71      address.Assign(p2p45.Install(nodes.Get(4), nodes.Get(5)));
72
73      //5-6
74      address.NewNetwork();
75      PointToPointHelper p2p56;
76      p2p56.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
77      p2p56.SetDeviceAttribute("DataRate",                                        ↵
        DataRateValue(DataRate(Case1_conf::DomainsLINKCAPACITY)));
78      p2p56.SetDeviceAttribute("InterframeGap",                                   ↵
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP)));
79      address.Assign(p2p56.Install(nodes.Get(5), nodes.Get(6)));
80
81      //6-8
82      address.NewNetwork();
83      PointToPointHelper p2p68;
84      p2p68.SetChannelAttribute("Delay",                                          ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY )));
85      p2p68.SetDeviceAttribute("DataRate",                                        ↵
        DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
86      p2p68.SetDeviceAttribute("InterframeGap",                                   ↵
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+2)));
87      address.Assign(p2p68.Install(nodes.Get(6), nodes.Get(8)));
88
89      //6-9
90      address.NewNetwork();
91      PointToPointHelper p2p69;
92      p2p69.SetChannelAttribute("Delay",                                          ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
93      p2p69.SetDeviceAttribute("DataRate",                                        ↵
        DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
94      p2p69.SetDeviceAttribute("InterframeGap",                                   ↵
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+2)));
95
96      address.Assign(p2p69.Install(nodes.Get(6), nodes.Get(9)));
97
98      //6-10
99      address.NewNetwork();
100     PointToPointHelper p2p610;
101     p2p610.SetChannelAttribute("Delay",                                         ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
102     p2p610.SetDeviceAttribute("DataRate",                                       ↵
        DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
103     p2p610.SetDeviceAttribute("InterframeGap",                                  ↵
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+2)));
104
```

```
105     address.Assign(p2p610.Install(nodes.Get(6), nodes.Get(10)));
106
107     //8-11
108     address.NewNetwork();
109     PointToPointHelper p2p811;
110     p2p811.SetChannelAttribute("Delay",                                          ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
111     p2p811.SetDeviceAttribute("DataRate",                                        ↵
        DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
112     p2p811.SetDeviceAttribute("InterframeGap",                                   ↵
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+4)));
113     address.Assign(p2p811.Install(nodes.Get(8), nodes.Get(11)));
114
115     //9-11
116     address.NewNetwork();
117     PointToPointHelper p2p911;
118     p2p911.SetChannelAttribute("Delay",                                          ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
119     p2p911.SetDeviceAttribute("DataRate",                                        ↵
        DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
120     p2p911.SetDeviceAttribute("InterframeGap",                                   ↵
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+4)));
121     address.Assign(p2p911.Install(nodes.Get(9), nodes.Get(11)));
122
123     //10-11
124     address.NewNetwork();
125     PointToPointHelper p2p1011;
126     p2p1011.SetChannelAttribute("Delay",                                         ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
127     p2p1011.SetDeviceAttribute("DataRate",                                       ↵
        DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
128     p2p1011.SetDeviceAttribute("InterframeGap",                                  ↵
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+4)));
129     address.Assign(p2p1011.Install(nodes.Get(10), nodes.Get(11)));
130
131     //8-15
132     address.NewNetwork();
133     PointToPointHelper p2p815;
134     p2p815.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
135     p2p815.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
136     address.Assign(p2p815.Install(nodes.Get(15), nodes.Get(8)));
137
138     //9-16
139     address.NewNetwork();
140     PointToPointHelper p2p916;
141     p2p916.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
142     p2p916.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
143     address.Assign(p2p916.Install(nodes.Get(16), nodes.Get(9)));
144
145     //10-17
146     address.NewNetwork();
147     PointToPointHelper p2p1017;
148     p2p1017.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
149     p2p1017.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
150     address.Assign(p2p1017.Install(nodes.Get(17), nodes.Get(10)));
151
152     //11-12
153     address.NewNetwork();
154     PointToPointHelper p2p1112;
155     p2p1112.SetChannelAttribute("Delay",                                         ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
```

```
156     p2p1112.SetDeviceAttribute("DataRate",
        DataRateValue(DataRate(Case1_conf::DestinationLINKCAPACITY)));
157     p2p1112.SetDeviceAttribute("InterframeGap",
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+6)));
158     address.Assign(p2p1112.Install(nodes.Get(11), nodes.Get(12)));
159
160     //11-13 Not used
161     address.NewNetwork();
162     PointToPointHelper p2p1113;
163     p2p1113.SetChannelAttribute("Delay",
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
164     p2p1113.SetDeviceAttribute("DataRate",
        DataRateValue(DataRate(Case1_conf::LINKCAPACITY)));
165     address.Assign(p2p1113.Install(nodes.Get(11), nodes.Get(13)));
166     address.NewNetwork();
167
168     //11-14 Not used
169     PointToPointHelper p2p1114;
170     p2p1114.SetChannelAttribute("Delay",
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
171     p2p1114.SetDeviceAttribute("DataRate",
        DataRateValue(DataRate(Case1_conf::LINKCAPACITY)));
172     address.Assign(p2p1114.Install(nodes.Get(11), nodes.Get(14)));
173     address.NewNetwork();
174
175     // 3-4
176     // notice: queue need to be set before install!!!
177     // you can't get the queue from outside the queue, so you need to update the
        parameter in the queue function.
178     // you need to set InterframeGap to decide the queue speed
179
180     queue234 = CreateObject<RDWQueue>(234);
181     queue234->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
        Case1_conf::SC3BYTEBUFFER);
182     queue234->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
        Case1_conf::REDSC3THIGH);
183     queue234->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
        Case1_conf::REDSC3HDROP);
184     queue234->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
        Case1_conf::REDSC3TLOW);
185     queue234->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
        Case1_conf::REDSC3LDROP);
186     queue234->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
187                         Case1_conf::SC2CONGESTIONFATORALPHAL,
188                         Case1_conf::SC3CONGESTIONFATORALPHAL);
189     queue234->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
190                         Case1_conf::SC2CONGESTIONFATORALPHAH,
191                         Case1_conf::SC3CONGESTIONFATORALPHAH);
192     queue234->setBELTA(
193         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
            Case1_conf::CONGESTIONFATBELTA3);
194
    nodes.Get(3)->GetDevice(4)->GetObject<PointToPointNetDevice>()->SetQueue(queue23
    4);
195
196     // 4-5
197
198     queue245 = CreateObject<RDWQueue>(245);
199     queue245->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
        Case1_conf::SC3BYTEBUFFER);
200     queue245->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
```

```
    Case1_conf::REDSC3THIGH);
    queue245->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
    Case1_conf::REDSC3HDROP);
    queue245->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
    Case1_conf::REDSC3TLOW);
    queue245->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
    Case1_conf::REDSC3LDROP);
    queue245->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
                        Case1_conf::SC2CONGESTIONFATORALPHAL,
                        Case1_conf::SC3CONGESTIONFATORALPHAL);
    queue245->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
                        Case1_conf::SC2CONGESTIONFATORALPHAH,
                        Case1_conf::SC3CONGESTIONFATORALPHAH);
    queue245->setBELTA(
        Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);

    nodes.Get(4)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue24
    5);

    // 5-6
    queue256 = CreateObject<RDWQueue>(256);
    queue256->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
    Case1_conf::SC3BYTEBUFFER);
    queue256->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
    Case1_conf::REDSC3THIGH);
    queue256->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
    Case1_conf::REDSC3HDROP);
    queue256->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
    Case1_conf::REDSC3TLOW);
    queue256->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
    Case1_conf::REDSC3LDROP);
    queue256->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
                        Case1_conf::SC2CONGESTIONFATORALPHAL,
                        Case1_conf::SC3CONGESTIONFATORALPHAL);
    queue256->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
                        Case1_conf::SC2CONGESTIONFATORALPHAH,
                        Case1_conf::SC3CONGESTIONFATORALPHAH);
    queue256->setBELTA(
        Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);

    nodes.Get(5)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue25
    6);

    // 6-8

    queue268 = CreateObject<RDWQueue>(268);
    queue268->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
    Case1_conf::SC3BYTEBUFFER);
    queue268->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
    Case1_conf::REDSC3THIGH);
    queue268->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
    Case1_conf::REDSC3HDROP);
    queue268->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
    Case1_conf::REDSC3TLOW);
    queue268->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
    Case1_conf::REDSC3LDROP);
    queue268->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
                        Case1_conf::SC2CONGESTIONFATORALPHAL,
                        Case1_conf::SC3CONGESTIONFATORALPHAL);
```

```
242     queue268->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
243                         Case1_conf::SC2CONGESTIONFATORALPHAH,
244                         Case1_conf::SC3CONGESTIONFATORALPHAH);
245     queue268->setBELTA(
246         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);
247
        nodes.Get(6)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue268);
248
249     // 6-9
250     queue269 = CreateObject<RDWQueue>(269);
251     queue269->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
        Case1_conf::SC3BYTEBUFFER);
252     queue269->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
        Case1_conf::REDSC3THIGH);
253     queue269->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
        Case1_conf::REDSC3HDROP);
254     queue269->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
        Case1_conf::REDSC3TLOW);
255     queue269->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
        Case1_conf::REDSC3LDROP);
256     queue269->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
257                         Case1_conf::SC2CONGESTIONFATORALPHAL,
258                         Case1_conf::SC3CONGESTIONFATORALPHAL);
259     queue269->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
260                         Case1_conf::SC2CONGESTIONFATORALPHAH,
261                         Case1_conf::SC3CONGESTIONFATORALPHAH);
262     queue269->setBELTA(
263         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);
264
        nodes.Get(6)->GetDevice(3)->GetObject<PointToPointNetDevice>()->SetQueue(queue269);
265
266     // 6-10
267
268     queue2610 = CreateObject<RDWQueue>(2610);
269     queue2610->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
        Case1_conf::SC3BYTEBUFFER);
270     queue2610->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
        Case1_conf::REDSC3THIGH);
271     queue2610->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
        Case1_conf::REDSC3HDROP);
272     queue2610->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
        Case1_conf::REDSC3TLOW);
273     queue2610->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
        Case1_conf::REDSC3LDROP);
274     queue2610->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
275                          Case1_conf::SC2CONGESTIONFATORALPHAL,
276                          Case1_conf::SC3CONGESTIONFATORALPHAL);
277     queue2610->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
278                          Case1_conf::SC2CONGESTIONFATORALPHAH,
279                          Case1_conf::SC3CONGESTIONFATORALPHAH);
280     queue2610->setBELTA(
281         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);
282
        nodes.Get(6)->GetDevice(4)->GetObject<PointToPointNetDevice>()->SetQueue(queue2610);
283     // 8-11
```

```
284       queue2811 = CreateObject<RDWQueue>(2811);
285       queue2811->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,  ↵
          Case1_conf::SC3BYTEBUFFER);
286       queue2811->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,        ↵
          Case1_conf::REDSC3THIGH);
287       queue2811->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,        ↵
          Case1_conf::REDSC3HDROP);
288       queue2811->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,           ↵
          Case1_conf::REDSC3TLOW);
289       queue2811->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,        ↵
          Case1_conf::REDSC3LDROP);
290       queue2811->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
291                            Case1_conf::SC2CONGESTIONFATORALPHAL,
292                            Case1_conf::SC3CONGESTIONFATORALPHAL);
293       queue2811->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
294                            Case1_conf::SC2CONGESTIONFATORALPHAH,
295                            Case1_conf::SC3CONGESTIONFATORALPHAH);
296       queue2811->setBELTA(
297           Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,            ↵
              Case1_conf::CONGESTIONFATBELTA3);
298                                                                                       ↵
          nodes.Get(8)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue28↵
          11);
299
300       // 9-11
301       queue2911 = CreateObject<RDWQueue>(2911);
302       queue2911->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,  ↵
          Case1_conf::SC3BYTEBUFFER);
303       queue2911->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,        ↵
          Case1_conf::REDSC3THIGH);
304       queue2911->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,        ↵
          Case1_conf::REDSC3HDROP);
305       queue2911->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,           ↵
          Case1_conf::REDSC3TLOW);
306       queue2911->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,        ↵
          Case1_conf::REDSC3LDROP);
307       queue2911->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
308                            Case1_conf::SC2CONGESTIONFATORALPHAL,
309                            Case1_conf::SC3CONGESTIONFATORALPHAL);
310       queue2911->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
311                            Case1_conf::SC2CONGESTIONFATORALPHAH,
312                            Case1_conf::SC3CONGESTIONFATORALPHAH);
313       queue2911->setBELTA(
314           Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,            ↵
              Case1_conf::CONGESTIONFATBELTA3);
315                                                                                       ↵
          nodes.Get(9)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue29↵
          11);
316
317       // 10-11
318       queue21011 = CreateObject<RDWQueue>(21011);
319       queue21011->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER,                            ↵
          Case1_conf::SC2BYTEBUFFER, Case1_conf::SC3BYTEBUFFER);
320       queue21011->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,       ↵
          Case1_conf::REDSC3THIGH);
321       queue21011->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,       ↵
          Case1_conf::REDSC3HDROP);
322       queue21011->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,          ↵
          Case1_conf::REDSC3TLOW);
323       queue21011->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,       ↵
          Case1_conf::REDSC3LDROP);
```

```
    queue21011->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
                          Case1_conf::SC2CONGESTIONFATORALPHAL,
                          Case1_conf::SC3CONGESTIONFATORALPHAL);
    queue21011->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
                          Case1_conf::SC2CONGESTIONFATORALPHAH,
                          Case1_conf::SC3CONGESTIONFATORALPHAH);
    queue21011->setBELTA(
        Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);

    nodes.Get(10)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue21011);

    // 11-12
    queue21112 = CreateObject<RDWQueue>(21112);
    queue21112->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER,
    Case1_conf::SC2BYTEBUFFER, Case1_conf::SC3BYTEBUFFER);
    queue21112->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
    Case1_conf::REDSC3THIGH);
    queue21112->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
    Case1_conf::REDSC3HDROP);
    queue21112->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
    Case1_conf::REDSC3TLOW);
    queue21112->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
    Case1_conf::REDSC3LDROP);
    queue21112->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
                          Case1_conf::SC2CONGESTIONFATORALPHAL,
                          Case1_conf::SC3CONGESTIONFATORALPHAL);
    queue21112->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
                          Case1_conf::SC2CONGESTIONFATORALPHAH,
                          Case1_conf::SC3CONGESTIONFATORALPHAH);
    queue21112->setBELTA(
        Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);

    nodes.Get(11)->GetDevice(4)->GetObject<PointToPointNetDevice>()->SetQueue(queue21112);

    for (uint32_t i = 0; i < nodes.GetN(); i++) {
        for (uint32_t j = 1; j < nodes.Get(i)->GetNDevices(); j++) {
            std::cout<<i<<" "<<nodes.Get(i)->GetObject<Ipv4> ()->GetAddress (j,
            0).GetLocal ()<<"Device"<<nodes.Get(i)->GetNDevices()<<"Get device
            index"<<nodes.Get(i) -> GetDevice(j)<<std::endl;
        }
    }

    Ipv4GlobalRoutingHelper::PopulateRoutingTables();

    // sink
    uint16_t sinkPort = 8080;

    PacketSinkHelper udpSinkHelper("ns3::UdpSocketFactory",
    InetSocketAddress(Ipv4Address::GetAny(), sinkPort));
    PacketSinkHelper tcpSinkHelper12tcp("ns3::TcpSocketFactory",
    InetSocketAddress(Ipv4Address::GetAny(), sinkPort));
    ApplicationContainer sinkApps12 = udpSinkHelper.Install(nodes.Get(12));
    sinkApps12.Start(Seconds(0.));
    ApplicationContainer sinkApps12tcp = tcpSinkHelper12tcp.Install(nodes.Get(12));
    sinkApps12tcp.Start(Seconds(0.));

    ApplicationContainer sinkApps13 = udpSinkHelper.Install(nodes.Get(13));
```

```
370       sinkApps13.Start(Seconds(0.));
371       ApplicationContainer sinkApps13tcp = tcpSinkHelper12tcp.Install(nodes.Get(13));
372       sinkApps13tcp.Start(Seconds(0.));
373
374       ApplicationContainer sinkApps14 = udpSinkHelper.Install(nodes.Get(14));
375       sinkApps14.Start(Seconds(0.));
376       ApplicationContainer sinkApps14tcp = tcpSinkHelper12tcp.Install(nodes.Get(14));
377       sinkApps14tcp.Start(Seconds(0.));
378
379       // SC1 traffic
380         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
381         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
382         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
383         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
384         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
385         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
386         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
387         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
388         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
389         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
390
391         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
392         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
393         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
394         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
395         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
396         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
397         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
398         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
399         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
400         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
401
402         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
403         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
404         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
405         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
          1);TrafficFromTo3(0, 12, 3, 3, 1);
406         TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                    ↵
```

```
        1);TrafficFromTo3(0, 12, 3, 3, 1);
407     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
408     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
409     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
410     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
411     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
412
413     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
414     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
415     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
416     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
417     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
418     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
419     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
420     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
421     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
422     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
423
424     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
425     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
426     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
427     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
428     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
429     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
430     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
431     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
432     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
433     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
434
435     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
436     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
437     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
        1);TrafficFromTo3(0, 12, 3, 3, 1);
438     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,                ↵
```

```
        1);TrafficFromTo3(0, 12, 3, 3, 1);
439     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,
        1);TrafficFromTo3(0, 12, 3, 3, 1);
440     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,
        1);TrafficFromTo3(0, 12, 3, 3, 1);
441     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,
        1);TrafficFromTo3(0, 12, 3, 3, 1);
442     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,
        1);TrafficFromTo3(0, 12, 3, 3, 1);
443     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,
        1);TrafficFromTo3(0, 12, 3, 3, 1);
444     TrafficFromTo3(0, 12, 3, 3, 1);TrafficFromTo3(0, 12, 3, 3,
        1);TrafficFromTo3(0, 12, 3, 3, 1);
445
446
447   // SC2 traffic
448     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
449     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
450     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
451     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
452     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
453     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
454     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
455     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
456     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
457     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
458
459     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
460     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
461     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
462     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
463     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
464     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
465     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
466     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
467     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
468     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
469
470     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
471     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
```

```
        2);TrafficFromTo3(0, 12, 4, 4, 2);
472     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
473     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
474     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
475     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
476     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
477     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
478     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
479     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
480
481     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
482     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
483     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
484     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
485     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
486     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
487     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
488     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
489     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
490     TrafficFromTo3(0, 12, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,
        2);TrafficFromTo3(0, 12, 4, 4, 2);
491
492
493 // SC3 traffic
494     TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
        3);TrafficFromTo3(0, 12, 5, 5, 3);
495     TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
        3);TrafficFromTo3(0, 12, 5, 5, 3);
496     TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
        3);TrafficFromTo3(0, 12, 5, 5, 3);
497     TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
        3);TrafficFromTo3(0, 12, 5, 5, 3);
498     TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
        3);TrafficFromTo3(0, 12, 5, 5, 3);
499     TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
        3);TrafficFromTo3(0, 12, 5, 5, 3);
500     TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
        3);TrafficFromTo3(0, 12, 5, 5, 3);
501     TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
        3);TrafficFromTo3(0, 12, 5, 5, 3);
502     TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
        3);TrafficFromTo3(0, 12, 5, 5, 3);
503     TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
        3);TrafficFromTo3(0, 12, 5, 5, 3);
```

```
504
505         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
506         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
507         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
508         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
509         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
510         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
511         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
512         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
513         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
514         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
515
516         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
517         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
518         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
519         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
520         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
521         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
522         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
523         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
524         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
525         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
526
527         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
528         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
529         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
530         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
531         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
532         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
533         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
534         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
535         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
```

```
536         TrafficFromTo3(0, 12, 5, 5, 3);TrafficFromTo3(0, 12, 5, 5,
            3);TrafficFromTo3(0, 12, 5, 5, 3);
537
538
539         // SC4 traffic
540     TrafficFromTo3(15, 12, 6, 6, 4); TrafficFromTo3(15, 12, 6, 6,
        4);TrafficFromTo3(15, 12, 6, 6, 4);
541     TrafficFromTo3(16, 12, 6, 6, 4); TrafficFromTo3(16, 12, 6, 6,
        4);TrafficFromTo3(16, 12, 6, 6, 4);
542     TrafficFromTo3(17, 12, 6, 6, 4); TrafficFromTo3(17, 12, 6, 6,
        4);TrafficFromTo3(17, 12, 6, 6, 4);
543
544
545
546     for (double i = 20000; i < Case1_conf::SIM_STOP_Sec * 1000; i += 1000) {
547         Simulator::Schedule(MilliSeconds(i), &Case2::ScheduleSchedulingRateUpdate,
            this);
548     }
549
550     for (double i = 10001; i < Case1_conf::SIM_STOP_Sec * 1000; i += 1000) {//1000
551
552         Simulator::Schedule(MilliSeconds(i), &Case2::AverageScheduleRate, this);
553
554     }
555
556
557     Config::ConnectWithoutContext("/NodeList/*/$ns3::Ipv4L3Protocol/Rx",
        MakeCallback(&MacRecv));
558 }
559
560 void Case2::TrafficFromTo1(int from, int to, int ProtocolType, int TrafficType,
    int ToS)
561 {
562     uint16_t sinkPort = 8080;
563     Address
        sinkAddress(InetSocketAddress(nodes.Get(to)->GetObject<Ipv4>()->GetAddress(1,
        0).GetLocal(), sinkPort));
564     Ptr<Socket> TrafficSocket;
565     Ptr<TosApp> app1 = CreateObject<TosApp>(ToS, TrafficType);
566     if (ProtocolType == 3) { // edit 13/05   VoIP traffic
567         TrafficSocket = Socket::CreateSocket(nodes.Get(from),
            UdpSocketFactory::GetTypeId());
568         app1->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (80000/1600)*Case1_conf::SIM_STOP_Sec, DataRate("80000"),1,1,1);
569
570     } else if (ProtocolType == 4) { // edit 13/05   Video Conferencing traffic
571         TrafficSocket = Socket::CreateSocket(nodes.Get(from),
            UdpSocketFactory::GetTypeId());
572         app1->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (384000/11200)*Case1_conf::SIM_STOP_Sec, DataRate("384000"),1,1,1);
573         // video conferencing rate 128-384 kbits/sec,
            https://en.wikipedia.org/wiki/Bit_rate
574     } else if (ProtocolType == 5) { // edit 13/05    (FTP) traffic  TcpSocketFactory
575         TrafficSocket =
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
576         app1->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (256000/2800)*Case1_conf::SIM_STOP_Sec, DataRate("256000"),1,1,1);
577
578     } else if (ProtocolType == 6) { // edit 13/05   // Database Query message
        traffic - Best effort traffic
579         TrafficSocket =
```

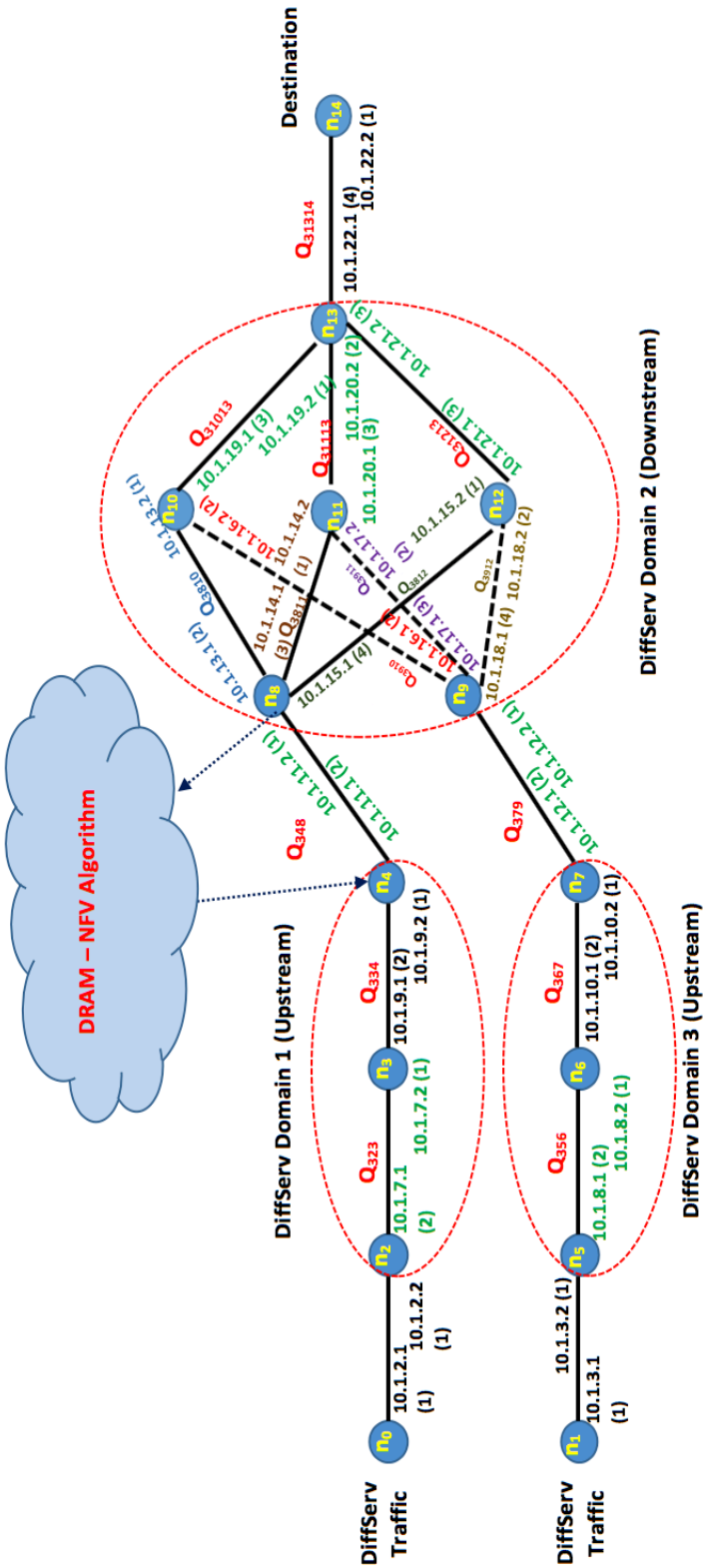
```
            Socket::CreateSocket(nodes.Get(from),UdpSocketFactory::GetTypeId());
580         app1->Setup(TrafficSocket, sinkAddress,                                    ↵
            Case1_conf::TRAFFICPKTSIZE,(81920/4096)*Case1_conf::SIM_STOP_Sec ,         ↵
            DataRate("81920"),1,1,1);
581
582     }
583     nodes.Get(from)->AddApplication(app1);
584
585     app1->SetStartTime(Seconds(1.));
586 }
587 void Case2::TrafficFromTo2(int from, int to, int ProtocolType, int TrafficType,    ↵
    int ToS)
588 {
589     uint16_t sinkPort = 8080;
590     Address                                                                        ↵
        sinkAddress(InetSocketAddress(nodes.Get(to)->GetObject<Ipv4>()->GetAddress(1,  ↵
        0).GetLocal(), sinkPort));
591     Ptr<Socket> TrafficSocket;
592     Ptr<TosApp> app2 = CreateObject<TosApp>(ToS, TrafficType);
593     if (ProtocolType == 3) { // edit 13/05   VoIP traffic
594         TrafficSocket = Socket::CreateSocket(nodes.Get(from),                      ↵
            UdpSocketFactory::GetTypeId());
595         app2->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,        ↵
            (80000/1600)*Case1_conf::SIM_STOP_Sec, DataRate("80000"),1,1,2);
596
597     } else if (ProtocolType == 4) { // edit 13/05   Video Conferencing traffic
598         TrafficSocket = Socket::CreateSocket(nodes.Get(from),                      ↵
            UdpSocketFactory::GetTypeId());
599         app2->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,        ↵
            (384000/11200)*Case1_conf::SIM_STOP_Sec, DataRate("384000"),1,1,2);
600         // video conferencing rate 128-384 kbits/sec,                              ↵
            https://en.wikipedia.org/wiki/Bit_rate
601     } else if (ProtocolType == 5) { // edit 13/05    (FTP) traffic  TcpSocketFactory
602         TrafficSocket =                                                            ↵
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
603         app2->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,        ↵
            (256000/2800)*Case1_conf::SIM_STOP_Sec, DataRate("256000"),1,1,2);
604
605     } else if (ProtocolType == 6) { // edit 13/05   // Database Query message      ↵
    traffic - Best effort traffic
606         TrafficSocket =                                                            ↵
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
607         app2->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,        ↵
            (81920/4096)*Case1_conf::SIM_STOP_Sec, DataRate("81920"),1,1,2);
608     }
609     nodes.Get(from)->AddApplication(app2);
610
611     app2->SetStartTime(Seconds(1.));
612 }
613 void Case2::TrafficFromTo3(int from, int to, int ProtocolType, int TrafficType,    ↵
    int ToS)
614 {
615     uint16_t sinkPort = 8080;
616     Address                                                                        ↵
        sinkAddress(InetSocketAddress(nodes.Get(to)->GetObject<Ipv4>()->GetAddress(1,  ↵
        0).GetLocal(), sinkPort));
617     Ptr<Socket> TrafficSocket;
618     Ptr<TosApp> app3 = CreateObject<TosApp>(ToS, TrafficType);
619     if (ProtocolType == 3) { // edit 13/05   VoIP traffic
620         TrafficSocket = Socket::CreateSocket(nodes.Get(from),                      ↵
            UdpSocketFactory::GetTypeId());
```

```
621         app3->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (80000/1600)*Case1_conf::SIM_STOP_Sec, DataRate("80000"),1,1,3);
622
623     } else if (ProtocolType == 4) { // edit 13/05   Video Conferencing traffic
624         TrafficSocket = Socket::CreateSocket(nodes.Get(from),
            UdpSocketFactory::GetTypeId());
625         app3->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (384000/11200)*Case1_conf::SIM_STOP_Sec, DataRate("384000"),1,1,3);
626         // video conferencing rate 128-384 kbits/sec,
            https://en.wikipedia.org/wiki/Bit_rate
627     } else if (ProtocolType == 5) { // edit 13/05    (FTP) traffic  TcpSocketFactory
628         TrafficSocket =
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
629         app3->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (256000/2800)*Case1_conf::SIM_STOP_Sec, DataRate("256000"),1,1,3);
630
631     }   else if (ProtocolType == 6) { // edit 13/05   // Database Query message
        traffic - Best effort traffic
632         TrafficSocket =
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
633         app3->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (81920/4096)*Case1_conf::SIM_STOP_Sec, DataRate("81920"),1,1,3);
634
635     }
636     nodes.Get(from)->AddApplication(app3);
637
638     app3->SetStartTime(Seconds(1.));
639
640
641 }
642
643 void Case2::ScheduleSchedulingRateUpdate(void)
644 {
645     double length68[4], length69[4], length610[4];
646     float AverageQueueLengthMaximum[4];
647     for (int i = 1; i <= 3; i++) {
648         length68[i] = queue268->GetAverageQueueLength1(i);
649         length69[i] = queue269->GetAverageQueueLength1(i);
650         length610[i] = queue2610->GetAverageQueueLength1(i);
651         AverageQueueLengthMaximum[i] = MaximumAverageQueueLength (length68[i],
            length69[i], length610[i]);
652         std::cout<<"length68:"<<" "<<length68[i]<<" "<<"length69:"<<"
            "<<length69[i]<<"  "<<"length610:"<<" "<<length610[i]<<" "<<i<<"
            "<<"maximum average length:"<<"  "<<AverageQueueLengthMaximum[i]<<std::endl;
653
654         }
655         // either section A or B work
656         // Section A
657    queue256->SetSchedulingRateByLinearFactor(1, AverageQueueLengthMaximum[1]);
658    queue256->SetSchedulingRateByLinearFactor(2, AverageQueueLengthMaximum[2]);
659    queue256->SetSchedulingRateByLinearFactor(3, AverageQueueLengthMaximum[3]);
660        // Section B
661   // queue256->SetSchedulingRateByFactor(1,
      AverageQueueLengthMaximum[1]/Case1_conf::SC1BYTEBUFFER);
662   // queue256->SetSchedulingRateByFactor(2,
      AverageQueueLengthMaximum[2]/Case1_conf::SC2BYTEBUFFER);
663   //  queue256->SetSchedulingRateByFactor(3,
      AverageQueueLengthMaximum[3]/Case1_conf::SC3BYTEBUFFER);
664
665 }
666
```

```
667
668 void Case2::AverageScheduleRate(void)
669 {
670     double Minqueuedelay [4],Maxqueuelength [4] ;
671     double length68[4], length69[4], length610[4];
672     double delay68[4], delay69[4], delay610[4];
673
674     for (int i = 1; i <= 3; i++) {
675         delay68[i] = queue268->GetAverageQueueDelay(i);
676         delay69[i] = queue269->GetAverageQueueDelay(i);
677         delay610[i] = queue2610->GetAverageQueueDelay(i);
678         Minqueuedelay [i] = MinimumAverageQueueDelay                                    ↵
          (delay68[i],delay69[i],delay610[i]);
679         std::cout<<"delay68:"<<" "<<delay68[i]<<" "<<"delay69:"<<"                      ↵
          "<<delay69[i]<<"  "<<"delay610:"<<" "<<delay610[i]<<" "<<i<<" "<<"minimum       ↵
          average delay:"<<"  "<<Minqueuedelay [i]<<std::endl;
680
681
682         length68[i] = queue268->GetAverageQueueLength2(i);
683         length69[i] = queue269->GetAverageQueueLength2(i);
684         length610[i] = queue2610->GetAverageQueueLength2(i);
685         Maxqueuelength [i] = MaximumAverageQueueLength(length68[i], length69[i],    ↵
          length610[i]);
686         std::cout<<"length68:"<<" "<<length68[i]<<" "<<"length69:"<<"               ↵
          "<<length69[i]<<"  "<<"length610:"<<" "<<length610[i]<<" "<<i<<"            ↵
          "<<"maximum average length:"<<"  "<<Maxqueuelength [i]<<std::endl;
687
688        queue268->SetSchedulingRateByAverage(i, Maxqueuelength [1], Maxqueuelength   ↵
          [2], Maxqueuelength [3], Minqueuedelay[1], Minqueuedelay[2],                 ↵
          Minqueuedelay[3]);
689        queue269->SetSchedulingRateByAverage(i, Maxqueuelength [1], Maxqueuelength   ↵
          [2], Maxqueuelength [3], Minqueuedelay[1], Minqueuedelay[2],                 ↵
          Minqueuedelay[3]);
690        queue2610->SetSchedulingRateByAverage(i, Maxqueuelength [1], Maxqueuelength  ↵
          [2], Maxqueuelength [3], Minqueuedelay[1], Minqueuedelay[2],                 ↵
          Minqueuedelay[3]);
691
692     }
693
694  }
695  double Case2::MinimumAverageQueueDelay(double x, double y, double z)
696 {    double minqueuedelay;
697     if (x !=0 && y !=0 && z!=0)
698         {
699          minqueuedelay = x;
700         if (y< minqueuedelay)
701         {
702         minqueuedelay = y;
703         }
704         if (z < minqueuedelay)
705         {
706         minqueuedelay = z;
707         }
708         }
709         else if (x ==0 && y !=0 && z !=0)
710         {
711          minqueuedelay = y;
712          if (z < minqueuedelay)
713          {
714          minqueuedelay = z;
715          }
```

```
716          }
717
718          else if (x !=0 && y ==0 && z !=0)
719          {
720           minqueuedelay = x;
721           if (z < minqueuedelay)
722           {
723           minqueuedelay = z;
724           }
725          }
726
727          else if (x !=0 && y !=0 && z ==0)
728          {
729           minqueuedelay = x;
730           if (y < minqueuedelay)
731           {
732           minqueuedelay = y;
733           }
734          }
735
736          else if (x ==0 && y ==0 && z !=0)
737          {
738           minqueuedelay = z;
739          }
740
741          else if (x ==0 && y !=0 && z ==0)
742          {
743           minqueuedelay = y;
744          }
745
746          else if (x !=0 && y ==0 && z ==0)
747          {
748           minqueuedelay = x;
749          }
750          else if (x ==0 && y ==0 && z ==0)
751          {
752           minqueuedelay = 0.1;
753          }
754          return minqueuedelay;
755  }
756  double Case2::MaximumAverageQueueLength(double x, double y, double z)
757  {
758      double maxqueuelebgth = x;
759         if (y>maxqueuelebgth)
760         {
761         maxqueuelebgth = y;
762         }
763         if (z > maxqueuelebgth)
764         {
765          maxqueuelebgth = z;
766         }
767          return maxqueuelebgth;
768  }
769
770  Case2::~Case2()
771  {
772      // dtor
773  }
774
```

A-5.3: Test Scenario 3, Topology Simulation Diagram and NS3 Code:



Queue	Scheduler	Queue	Scheduler
Q323	Dynamic	Q31213	Fixed
Q334	Fixed	Q31314	Dynamic
Q356	Dynamic	Q3810	Dynamic such that the scheduling rate of SC¡3810=SC¡3811=SC¡3812= SC¡3910=SC¡3911=SC¡3912
Q367	Fixed	Q3811	Dynamic such that the scheduling rate of SC¡3810=SC¡3811=SC¡3812= SC¡3910=SC¡3911=SC¡3912
Q348	Dynamic	Q3812	Dynamic such that the scheduling rate of SC¡3810=SC¡3811=SC¡3812= SC¡3910=SC¡3911=SC¡3912
Q379	Dynamic	Q3910	Dynamic such that the scheduling rate of SC¡3810=SC¡3811=SC¡3812= SC¡3910=SC¡3911=SC¡3912
Q31013	Fixed	Q3911	Dynamic such that the scheduling rate of SC¡3810=SC¡3811=SC¡3812= SC¡3910=SC¡3911=SC¡3912
Q31113	Fixed	Q3912	Dynamic such that the scheduling rate of SC¡3810=SC¡3811=SC¡3812= SC¡3910=SC¡3911=SC¡3912

Figure 9-10, test scenario 3 simulation network topology diagram.

```cpp
#ifndef CASE3_H
#define CASE3_H

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/csma-module.h"
#include "RDWQueue.h"

#include "ns3/Case1_conf.h"


using namespace ns3;

class Case3
{
    public:
        Case3();
        virtual ~Case3();

    NodeContainer nodes;

   Ptr<RDWQueue> queue323;
   Ptr<RDWQueue> queue334;
   Ptr<RDWQueue> queue356;
   Ptr<RDWQueue> queue367;
   Ptr<RDWQueue> queue348;
   Ptr<RDWQueue> queue379;
   Ptr<RDWQueue> queue3810;
   Ptr<RDWQueue> queue3811;
   Ptr<RDWQueue> queue3812;
   Ptr<RDWQueue> queue3910;
   Ptr<RDWQueue> queue3911;
   Ptr<RDWQueue> queue3912;
   Ptr<RDWQueue> queue31013;
   Ptr<RDWQueue> queue31113;
   Ptr<RDWQueue> queue31213;
   Ptr<RDWQueue> queue31314;

   void ScheduleSchedulingRateUpdate(void);
   //void TrafficFromTo(int from, int to, int ProtocolType ,int TrafficType, int ToS);
   void TrafficFromTo1(int from, int to, int ProtocolType ,int TrafficType, int ToS);
   void TrafficFromTo2(int from, int to, int ProtocolType ,int TrafficType, int ToS);
   void TrafficFromTo3(int from, int to, int ProtocolType ,int TrafficType, int ToS);
   void AverageScheduleRate(void);
   double MinimumAverageQueueDelay(double x, double y, double z);
   double MaximumAverageQueueLength(double x, double y, double z);


};

#endif // CASE3_H
```

```cpp

#include "Case3.h"
#include "TosApp.h"
#include "Case1_conf.h"


using namespace ns3;

// static void MacRecv(Ptr<const Packet> packet)
static void MacRecv(const Ptr<const Packet> packet, Ptr<Ipv4> ipv4, const uint32_t interface)
{
    Ipv4Header header;
    packet->PeekHeader(header);
    if (header.GetDestination().IsEqual(ipv4->GetAddress(interface, 0).GetLocal())) {
        QosTag tosTag;
        uint32_t band = 0;
        if (packet->PeekPacketTag(tosTag)) {
            band = (uint32_t)tosTag.GetTid();
        }
        if (band != 0 ) {
            std::cout << "Pkt_recv " << packet->GetUid() << " " << Simulator::Now() << " " << band << std::endl;
        }
    }
}

Case3::Case3()
{
    Config::SetDefault("ns3::Ipv4GlobalRouting::RandomEcmpRouting", BooleanValue(true));
    nodes.Create(18);

    InternetStackHelper stack;
    stack.Install(nodes);

    Ipv4AddressHelper address;
    address.SetBase("10.1.1.0", "255.255.255.0");
////////////////Sceniro 3 ////////////////////
//0-2
    address.NewNetwork();
    PointToPointHelper p2p02;
    p2p02.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
    p2p02.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
    address.Assign(p2p02.Install(nodes.Get(0), nodes.Get(2)));

//1-5
    address.NewNetwork();
    PointToPointHelper p2p15;
    p2p15.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
    p2p15.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
    address.Assign(p2p15.Install(nodes.Get(1), nodes.Get(5)));

//15-10
    address.NewNetwork();
    PointToPointHelper p2p1510;
    p2p1510.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
    p2p1510.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
    address.Assign(p2p1510.Install(nodes.Get(15), nodes.Get(10)));

//16-11
```

```
59      address.NewNetwork();
60      PointToPointHelper p2p1511;
61      p2p1511.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
62      p2p1511.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
63      address.Assign(p2p1511.Install(nodes.Get(16), nodes.Get(11)));
64
65  //17-12
66      address.NewNetwork();
67      PointToPointHelper p2p1512;
68      p2p1512.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
69      p2p1512.SetDeviceAttribute("DataRate", DataRateValue(DataRate(1000000000)));
70      address.Assign(p2p1512.Install(nodes.Get(17), nodes.Get(12)));
71
72  //2-3
73      address.NewNetwork();
74      PointToPointHelper p2p23;
75      p2p23.SetChannelAttribute("Delay",                                          ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
76      p2p23.SetDeviceAttribute("DataRate",                                        ↵
        DataRateValue(DataRate(Case1_conf::UpstreamLINKCAPACITY)));
77      p2p23.SetDeviceAttribute("InterframeGap", TimeValue(MilliSeconds(0)));
78      address.Assign(p2p23.Install(nodes.Get(2), nodes.Get(3)));
79
80      queue323 = CreateObject<RDWQueue>(323);
81      queue323->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,  ↵
        Case1_conf::SC3BYTEBUFFER);
82      queue323->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,       ↵
        Case1_conf::REDSC3THIGH);
83      queue323->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,       ↵
        Case1_conf::REDSC3HDROP);
84      queue323->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,          ↵
        Case1_conf::REDSC3TLOW);
85      queue323->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,       ↵
        Case1_conf::REDSC3LDROP);
86      queue323->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
87                          Case1_conf::SC2CONGESTIONFATORALPHAL,
88                          Case1_conf::SC3CONGESTIONFATORALPHAL);
89      queue323->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
90                          Case1_conf::SC2CONGESTIONFATORALPHAH,
91                          Case1_conf::SC3CONGESTIONFATORALPHAH);
92      queue323->setBELTA(
93          Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,        ↵
            Case1_conf::CONGESTIONFATBELTA3);
94
95                                                                                  ↵
        nodes.Get(2)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue32↵
        3);
96
97
98  //5-6
99      address.NewNetwork();
100     PointToPointHelper p2p56;
101     p2p56.SetChannelAttribute("Delay",                                          ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
102     p2p56.SetDeviceAttribute("DataRate",                                        ↵
        DataRateValue(DataRate(Case1_conf::UpstreamLINKCAPACITY)));
103     p2p56.SetDeviceAttribute("InterframeGap", TimeValue(MilliSeconds(0)));
104     address.Assign(p2p56.Install(nodes.Get(5), nodes.Get(6)));
105
106     queue356 = CreateObject<RDWQueue>(356);
107     queue356->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,  ↵
```

```
    Case1_conf::SC3BYTEBUFFER);
    queue356->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
    Case1_conf::REDSC3THIGH);
    queue356->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
    Case1_conf::REDSC3HDROP);
    queue356->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
    Case1_conf::REDSC3TLOW);
    queue356->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
    Case1_conf::REDSC3LDROP);
    queue356->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
                        Case1_conf::SC2CONGESTIONFATORALPHAL,
                        Case1_conf::SC3CONGESTIONFATORALPHAL);
    queue356->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
                        Case1_conf::SC2CONGESTIONFATORALPHAH,
                        Case1_conf::SC3CONGESTIONFATORALPHAH);
    queue356->setBELTA(Case1_conf::CONGESTIONFATBELTA1,
    Case1_conf::CONGESTIONFATBELTA2, Case1_conf::CONGESTIONFATBELTA3);

    nodes.Get(5)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue35
    6);

//3-4
    address.NewNetwork();
    PointToPointHelper p2p34;
    p2p34.SetChannelAttribute("Delay",
    TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
    p2p34.SetDeviceAttribute("DataRate",
    DataRateValue(DataRate(Case1_conf::UpstreamLINKCAPACITY)));
    p2p34.SetDeviceAttribute("InterframeGap", TimeValue(MilliSeconds(0.5)));
    address.Assign(p2p34.Install(nodes.Get(3), nodes.Get(4)));

    queue334 = CreateObject<RDWQueue>(334);
    queue334->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
    Case1_conf::SC3BYTEBUFFER);
    queue334->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
    Case1_conf::REDSC3THIGH);
    queue334->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
    Case1_conf::REDSC3HDROP);
    queue334->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
    Case1_conf::REDSC3TLOW);
    queue334->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
    Case1_conf::REDSC3LDROP);
    queue334->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
                        Case1_conf::SC2CONGESTIONFATORALPHAL,
                        Case1_conf::SC3CONGESTIONFATORALPHAL);
    queue334->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
                        Case1_conf::SC2CONGESTIONFATORALPHAH,
                        Case1_conf::SC3CONGESTIONFATORALPHAH);
    queue334->setBELTA(
        Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);

    nodes.Get(3)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue33
    4);

//6-7
    address.NewNetwork();
    PointToPointHelper p2p67;
    p2p67.SetChannelAttribute("Delay",
    TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
    p2p67.SetDeviceAttribute("DataRate",
```

```
    DataRateValue(DataRate(Case1_conf::UpstreamLINKCAPACITY)));
    p2p67.SetDeviceAttribute("InterframeGap", TimeValue(MilliSeconds(0.5)));
    address.Assign(p2p67.Install(nodes.Get(6), nodes.Get(7)));

    queue367 = CreateObject<RDWQueue>(367);
    queue367->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
    Case1_conf::SC3BYTEBUFFER);
    queue367->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
    Case1_conf::REDSC3THIGH);
    queue367->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
    Case1_conf::REDSC3HDROP);
    queue367->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
    Case1_conf::REDSC3TLOW);
    queue367->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
    Case1_conf::REDSC3LDROP);
    queue367->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
                        Case1_conf::SC2CONGESTIONFATORALPHAL,
                        Case1_conf::SC3CONGESTIONFATORALPHAL);
    queue367->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
                        Case1_conf::SC2CONGESTIONFATORALPHAH,
                        Case1_conf::SC3CONGESTIONFATORALPHAH);
    queue367->setBELTA(
        Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);

    nodes.Get(6)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue36
    7);

//4-8
    address.NewNetwork();
    PointToPointHelper p2p48;
    p2p48.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
    p2p48.SetDeviceAttribute("DataRate",
    DataRateValue(DataRate(Case1_conf::DomainsLINKCAPACITY)));
    p2p48.SetDeviceAttribute("InterframeGap", TimeValue(MilliSeconds(1)));
    address.Assign(p2p48.Install(nodes.Get(4), nodes.Get(8)));

    queue348 = CreateObject<RDWQueue>(348);
    queue348->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
    Case1_conf::SC3BYTEBUFFER);
    queue348->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
    Case1_conf::REDSC3THIGH);
    queue348->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
    Case1_conf::REDSC3HDROP);
    queue348->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
    Case1_conf::REDSC3TLOW);
    queue348->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
    Case1_conf::REDSC3LDROP);
    queue348->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
                        Case1_conf::SC2CONGESTIONFATORALPHAL,
                        Case1_conf::SC3CONGESTIONFATORALPHAL);
    queue348->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
                        Case1_conf::SC2CONGESTIONFATORALPHAH,
                        Case1_conf::SC3CONGESTIONFATORALPHAH);
    queue348->setBELTA(
        Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);

    nodes.Get(4)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue34
    8);

```

```
193 //7-9
194     address.NewNetwork();
195     PointToPointHelper p2p79;
196     p2p79.SetChannelAttribute("Delay", TimeValue(MilliSeconds(0.1)));
197     p2p79.SetDeviceAttribute("DataRate",                                    ↵
        DataRateValue(DataRate(Case1_conf::DomainsLINKCAPACITY)));
198     p2p79.SetDeviceAttribute("InterframeGap", TimeValue(MilliSeconds(1)));
199     address.Assign(p2p79.Install(nodes.Get(7), nodes.Get(9)));
200
201    queue379 = CreateObject<RDWQueue>(379);
202     queue379->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,  ↵
        Case1_conf::SC3BYTEBUFFER);
203     queue379->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,       ↵
        Case1_conf::REDSC3THIGH);
204     queue379->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,       ↵
        Case1_conf::REDSC3HDROP);
205     queue379->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,          ↵
        Case1_conf::REDSC3TLOW);
206     queue379->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,       ↵
        Case1_conf::REDSC3LDROP);
207     queue379->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
208                         Case1_conf::SC2CONGESTIONFATORALPHAL,
209                         Case1_conf::SC3CONGESTIONFATORALPHAL);
210     queue379->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
211                         Case1_conf::SC2CONGESTIONFATORALPHAH,
212                         Case1_conf::SC3CONGESTIONFATORALPHAH);
213     queue379->setBELTA(
214         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,          ↵
            Case1_conf::CONGESTIONFATBELTA3);
215                                                                                   ↵
        nodes.Get(7)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue37↵
        9);
216
217 //8-10
218     address.NewNetwork();
219     PointToPointHelper p2p3810;
220     p2p3810.SetChannelAttribute("Delay",                                      ↵
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
221     p2p3810.SetDeviceAttribute("DataRate",                                    ↵
        DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
222     p2p3810.SetDeviceAttribute("InterframeGap",                               ↵
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+2)));
223     address.Assign(p2p3810.Install(nodes.Get(8), nodes.Get(10)));
224
225     queue3810 = CreateObject<RDWQueue>(3810);
226     queue3810->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,  ↵
        Case1_conf::SC3BYTEBUFFER);
227     queue3810->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,      ↵
        Case1_conf::REDSC3THIGH);
228     queue3810->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,      ↵
        Case1_conf::REDSC3HDROP);
229     queue3810->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,         ↵
        Case1_conf::REDSC3TLOW);
230     queue3810->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,      ↵
        Case1_conf::REDSC3LDROP);
231     queue3810->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
232                         Case1_conf::SC2CONGESTIONFATORALPHAL,
233                         Case1_conf::SC3CONGESTIONFATORALPHAL);
234     queue3810->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
235                         Case1_conf::SC2CONGESTIONFATORALPHAH,
236                         Case1_conf::SC3CONGESTIONFATORALPHAH);
```

```
237     queue3810->setBELTA(
238         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);
239
        nodes.Get(8)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue3810);
240
241 //8-11
242     address.NewNetwork();
243     PointToPointHelper p2p3811;
244     p2p3811.SetChannelAttribute("Delay",
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
245     p2p3811.SetDeviceAttribute("DataRate",
        DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
246     p2p3811.SetDeviceAttribute("InterframeGap",
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+2)));
247     address.Assign(p2p3811.Install(nodes.Get(8), nodes.Get(11)));
248
249     queue3811 = CreateObject<RDWQueue>(3811);
250     queue3811->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
        Case1_conf::SC3BYTEBUFFER);
251     queue3811->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
        Case1_conf::REDSC3THIGH);
252     queue3811->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
        Case1_conf::REDSC3HDROP);
253     queue3811->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
        Case1_conf::REDSC3TLOW);
254     queue3811->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
        Case1_conf::REDSC3LDROP);
255     queue3811->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
256                          Case1_conf::SC2CONGESTIONFATORALPHAL,
257                          Case1_conf::SC3CONGESTIONFATORALPHAL);
258     queue3811->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
259                          Case1_conf::SC2CONGESTIONFATORALPHAH,
260                          Case1_conf::SC3CONGESTIONFATORALPHAH);
261     queue3811->setBELTA(
262         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);
263
        nodes.Get(8)->GetDevice(3)->GetObject<PointToPointNetDevice>()->SetQueue(queue3811);
264
265 //8-12
266     address.NewNetwork();
267     PointToPointHelper p2p3812;
268     p2p3812.SetChannelAttribute("Delay",
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
269     p2p3812.SetDeviceAttribute("DataRate",
        DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
270     p2p3812.SetDeviceAttribute("InterframeGap",
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+2)));
271     address.Assign(p2p3812.Install(nodes.Get(8), nodes.Get(12)));
272
273     queue3812 = CreateObject<RDWQueue>(3812);
274     queue3812->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
        Case1_conf::SC3BYTEBUFFER);
275     queue3812->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
        Case1_conf::REDSC3THIGH);
276     queue3812->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
        Case1_conf::REDSC3HDROP);
277     queue3812->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
```

```
    Case1_conf::REDSC3TLOW);
278     queue3812->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
    Case1_conf::REDSC3LDROP);
279     queue3812->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
280                          Case1_conf::SC2CONGESTIONFATORALPHAL,
281                          Case1_conf::SC3CONGESTIONFATORALPHAL);
282     queue3812->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
283                          Case1_conf::SC2CONGESTIONFATORALPHAH,
284                          Case1_conf::SC3CONGESTIONFATORALPHAH);
285     queue3812->setBELTA(
286         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);
287
    nodes.Get(8)->GetDevice(4)->GetObject<PointToPointNetDevice>()->SetQueue(queue38
    12);
288
289 //9-10
290     address.NewNetwork();
291     PointToPointHelper p2p3910;
292     p2p3910.SetChannelAttribute("Delay",
    TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
293     p2p3910.SetDeviceAttribute("DataRate",
    DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
294     p2p3910.SetDeviceAttribute("InterframeGap",
    TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+2)));
295     address.Assign(p2p3910.Install(nodes.Get(9), nodes.Get(10)));
296
297     queue3910 = CreateObject<RDWQueue>(3910);
298     queue3910->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
    Case1_conf::SC3BYTEBUFFER);
299     queue3910->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
    Case1_conf::REDSC3THIGH);
300     queue3910->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
    Case1_conf::REDSC3HDROP);
301     queue3910->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
    Case1_conf::REDSC3TLOW);
302     queue3910->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
    Case1_conf::REDSC3LDROP);
303     queue3910->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
304                          Case1_conf::SC2CONGESTIONFATORALPHAL,
305                          Case1_conf::SC3CONGESTIONFATORALPHAL);
306     queue3910->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
307                          Case1_conf::SC2CONGESTIONFATORALPHAH,
308                          Case1_conf::SC3CONGESTIONFATORALPHAH);
309     queue3910->setBELTA(
310         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);
311
    nodes.Get(9)->GetDevice(2)->GetObject<PointToPointNetDevice>()->SetQueue(queue39
    10);
312
313 //9-11
314     address.NewNetwork();
315     PointToPointHelper p2p3911;
316     p2p3911.SetChannelAttribute("Delay",
    TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
317     p2p3911.SetDeviceAttribute("DataRate",
    DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
318     p2p3911.SetDeviceAttribute("InterframeGap",
    TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+2)));
319     address.Assign(p2p3911.Install(nodes.Get(9), nodes.Get(11)));
```

```
320
321     queue3911 = CreateObject<RDWQueue>(3911);
322     queue3911->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
        Case1_conf::SC3BYTEBUFFER);
323     queue3911->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
        Case1_conf::REDSC3THIGH);
324     queue3911->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
        Case1_conf::REDSC3HDROP);
325     queue3911->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
        Case1_conf::REDSC3TLOW);
326     queue3911->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
        Case1_conf::REDSC3LDROP);
327     queue3911->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
328                          Case1_conf::SC2CONGESTIONFATORALPHAL,
329                          Case1_conf::SC3CONGESTIONFATORALPHAL);
330     queue3911->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
331                          Case1_conf::SC2CONGESTIONFATORALPHAH,
332                          Case1_conf::SC3CONGESTIONFATORALPHAH);
333     queue3911->setBELTA(
334         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
            Case1_conf::CONGESTIONFATBELTA3);
335
        nodes.Get(9)->GetDevice(3)->GetObject<PointToPointNetDevice>()->SetQueue(queue39
        11);
336
337 //9-12
338     address.NewNetwork();
339     PointToPointHelper p2p3912;
340     p2p3912.SetChannelAttribute("Delay",
        TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
341     p2p3912.SetDeviceAttribute("DataRate",
        DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
342     p2p3912.SetDeviceAttribute("InterframeGap",
        TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+2)));
343     address.Assign(p2p3912.Install(nodes.Get(9), nodes.Get(12)));
344
345     queue3912 = CreateObject<RDWQueue>(3912);
346     queue3912->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER, Case1_conf::SC2BYTEBUFFER,
        Case1_conf::SC3BYTEBUFFER);
347     queue3912->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
        Case1_conf::REDSC3THIGH);
348     queue3912->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
        Case1_conf::REDSC3HDROP);
349     queue3912->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
        Case1_conf::REDSC3TLOW);
350     queue3912->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
        Case1_conf::REDSC3LDROP);
351     queue3912->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
352                          Case1_conf::SC2CONGESTIONFATORALPHAL,
353                          Case1_conf::SC3CONGESTIONFATORALPHAL);
354     queue3912->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
355                          Case1_conf::SC2CONGESTIONFATORALPHAH,
356                          Case1_conf::SC3CONGESTIONFATORALPHAH);
357     queue3912->setBELTA(
358         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
            Case1_conf::CONGESTIONFATBELTA3);
359
        nodes.Get(9)->GetDevice(4)->GetObject<PointToPointNetDevice>()->SetQueue(queue39
        12);
360
361 //10-13
```

```
362       address.NewNetwork();
363       PointToPointHelper p2p1013;
364       p2p1013.SetChannelAttribute("Delay",                                    ↵
          TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
365       p2p1013.SetDeviceAttribute("DataRate",                                  ↵
          DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
366
367       p2p1013.SetDeviceAttribute("InterframeGap",                             ↵
          TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+4)));
368       address.Assign(p2p1013.Install(nodes.Get(10), nodes.Get(13)));
369
370       queue31013 = CreateObject<RDWQueue>(31013);
371       queue31013->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER,                    ↵
          Case1_conf::SC2BYTEBUFFER, Case1_conf::SC3BYTEBUFFER);
372       queue31013->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,  ↵
          Case1_conf::REDSC3THIGH);
373       queue31013->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,  ↵
          Case1_conf::REDSC3HDROP);
374       queue31013->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,     ↵
          Case1_conf::REDSC3TLOW);
375       queue31013->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,  ↵
          Case1_conf::REDSC3LDROP);
376       queue31013->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
377                             Case1_conf::SC2CONGESTIONFATORALPHAL,
378                             Case1_conf::SC3CONGESTIONFATORALPHAL);
379       queue31013->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
380                             Case1_conf::SC2CONGESTIONFATORALPHAH,
381                             Case1_conf::SC3CONGESTIONFATORALPHAH);
382       queue31013->setBELTA(
383           Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,    ↵
              Case1_conf::CONGESTIONFATBELTA3);
384                                                                               ↵
          nodes.Get(10)->GetDevice(4)->GetObject<PointToPointNetDevice>()->SetQueue(queue3↵
          1013);
385
386   //11-13
387       address.NewNetwork();
388       PointToPointHelper p2p1113;
389       p2p1113.SetChannelAttribute("Delay",                                    ↵
          TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
390       p2p1113.SetDeviceAttribute("DataRate",                                  ↵
          DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
391
392       p2p1113.SetDeviceAttribute("InterframeGap",                             ↵
          TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+4)));
393       address.Assign(p2p1113.Install(nodes.Get(11), nodes.Get(13)));
394
395       queue31113 = CreateObject<RDWQueue>(31113);
396       queue31113->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER,                    ↵
          Case1_conf::SC2BYTEBUFFER, Case1_conf::SC3BYTEBUFFER);
397       queue31113->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,  ↵
          Case1_conf::REDSC3THIGH);
398       queue31113->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,  ↵
          Case1_conf::REDSC3HDROP);
399       queue31113->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,     ↵
          Case1_conf::REDSC3TLOW);
400       queue31113->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,  ↵
          Case1_conf::REDSC3LDROP);
401       queue31113->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
402                             Case1_conf::SC2CONGESTIONFATORALPHAL,
403                             Case1_conf::SC3CONGESTIONFATORALPHAL);
```

```
404     queue31113->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
405                           Case1_conf::SC2CONGESTIONFATORALPHAH,
406                           Case1_conf::SC3CONGESTIONFATORALPHAH);
407     queue31113->setBELTA(
408         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);
409
    nodes.Get(10)->GetDevice(4)->GetObject<PointToPointNetDevice>()->SetQueue(queue3
    1113);
410
411 //12-13
412     address.NewNetwork();
413     PointToPointHelper p2p1213;
414     p2p1213.SetChannelAttribute("Delay",
    TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
415     p2p1213.SetDeviceAttribute("DataRate",
    DataRateValue(DataRate(Case1_conf::DownstreamLINKCAPACITY)));
416
417     p2p1213.SetDeviceAttribute("InterframeGap",
    TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+4)));
418
419     address.Assign(p2p1213.Install(nodes.Get(12), nodes.Get(13)));
420
421     queue31213 = CreateObject<RDWQueue>(31213);
422     queue31213->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER,
    Case1_conf::SC2BYTEBUFFER, Case1_conf::SC3BYTEBUFFER);
423     queue31213->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
    Case1_conf::REDSC3THIGH);
424     queue31213->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
    Case1_conf::REDSC3HDROP);
425     queue31213->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
    Case1_conf::REDSC3TLOW);
426     queue31213->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
    Case1_conf::REDSC3LDROP);
427     queue31213->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
428                           Case1_conf::SC2CONGESTIONFATORALPHAL,
429                           Case1_conf::SC3CONGESTIONFATORALPHAL);
430     queue31213->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
431                           Case1_conf::SC2CONGESTIONFATORALPHAH,
432                           Case1_conf::SC3CONGESTIONFATORALPHAH);
433     queue31213->setBELTA(
434         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);
435
    nodes.Get(12)->GetDevice(4)->GetObject<PointToPointNetDevice>()->SetQueue(queue3
    1213);
436
437 //13-14
438     address.NewNetwork();
439     PointToPointHelper p2p1314;
440     p2p1314.SetChannelAttribute("Delay",
    TimeValue(MilliSeconds(Case1_conf::LINKDELAY)));
441     p2p1314.SetDeviceAttribute("DataRate",
    DataRateValue(DataRate(Case1_conf::DestinationLINKCAPACITY)));
442     p2p1314.SetDeviceAttribute("InterframeGap",
    TimeValue(MilliSeconds(Case1_conf::INTERFRAMEGAP+6)));
443     address.Assign(p2p1314.Install(nodes.Get(13), nodes.Get(14)));
444
445     queue31314 = CreateObject<RDWQueue>(31314);
446     queue31314->setBYTEBUFFER(Case1_conf::SC1BYTEBUFFER,
    Case1_conf::SC2BYTEBUFFER, Case1_conf::SC3BYTEBUFFER);
```

```
447     queue31314->setREDTHIGH(Case1_conf::REDSC1THIGH, Case1_conf::REDSC2THIGH,
        Case1_conf::REDSC3THIGH);
448     queue31314->setREDHDROP(Case1_conf::REDSC1HDROP, Case1_conf::REDSC2HDROP,
        Case1_conf::REDSC3HDROP);
449     queue31314->setREDTLOW(Case1_conf::REDSC1TLOW, Case1_conf::REDSC2TLOW,
        Case1_conf::REDSC3TLOW);
450     queue31314->setREDLDROP(Case1_conf::REDSC1LDROP, Case1_conf::REDSC2LDROP,
        Case1_conf::REDSC3LDROP);
451     queue31314->setALPHAL(Case1_conf::SC1CONGESTIONFATORALPHAL,
452                           Case1_conf::SC2CONGESTIONFATORALPHAL,
453                           Case1_conf::SC3CONGESTIONFATORALPHAL);
454     queue31314->setALPHAH(Case1_conf::SC1CONGESTIONFATORALPHAH,
455                           Case1_conf::SC2CONGESTIONFATORALPHAH,
456                           Case1_conf::SC3CONGESTIONFATORALPHAH);
457     queue31314->setBELTA(
458         Case1_conf::CONGESTIONFATBELTA1, Case1_conf::CONGESTIONFATBELTA2,
        Case1_conf::CONGESTIONFATBELTA3);
459
        nodes.Get(13)->GetDevice(4)->GetObject<PointToPointNetDevice>()->SetQueue(queue3
        1314);
460
461
462
463     for (uint32_t i = 0; i < nodes.GetN(); i++) {
464         for (uint32_t j = 1; j < nodes.Get(i)->GetNDevices(); j++) {
465          std::cout<<i<<" "<<nodes.Get(i)->GetObject<Ipv4> ()->GetAddress (j,
            0).GetLocal ()<<"Device"<<nodes.Get(i)->GetNDevices()<<"Get device
            index"<<nodes.Get(i) -> GetDevice(j)<<std::endl;
466         }
467     }
468
469     Ipv4GlobalRoutingHelper::PopulateRoutingTables();
470
471 // sink
472     uint16_t sinkPort = 8080;
473
474     PacketSinkHelper udpSinkHelper("ns3::UdpSocketFactory",
        InetSocketAddress(Ipv4Address::GetAny(), sinkPort));
475     PacketSinkHelper tcpSinkHelper14tcp("ns3::TcpSocketFactory",
        InetSocketAddress(Ipv4Address::GetAny(), sinkPort));
476     ApplicationContainer sinkApps14 = udpSinkHelper.Install(nodes.Get(14));
477     sinkApps14.Start(Seconds(0.));
478     ApplicationContainer sinkApps14tcp = tcpSinkHelper14tcp.Install(nodes.Get(14));
479     sinkApps14tcp.Start(Seconds(0.));
480
481     // Traffic 0
482      TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
        1);TrafficFromTo3(0, 14, 3, 3, 1);
483      TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
        1);TrafficFromTo3(0, 14, 3, 3, 1);
484      TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
        1);TrafficFromTo3(0, 14, 3, 3, 1);
485      TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
        1);TrafficFromTo3(0, 14, 3, 3, 1);
486      TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
        1);TrafficFromTo3(0, 14, 3, 3, 1);
487      TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
        1);TrafficFromTo3(0, 14, 3, 3, 1);
488      TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
        1);TrafficFromTo3(0, 14, 3, 3, 1);
489      TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
```

```
      1);TrafficFromTo3(0, 14, 3, 3, 1);
490   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
491   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
492
493   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
494   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
495   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
496   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
497   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
498   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
499   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
500   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
501   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
502   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
503
504   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
505   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
506   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
507   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
508   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
509   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
510   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
511   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
512   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
513   TrafficFromTo3(0, 14, 3, 3, 1);TrafficFromTo3(0, 14, 3, 3,
      1);TrafficFromTo3(0, 14, 3, 3, 1);
514
515   TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,
      2);TrafficFromTo3(0, 14, 4, 4, 2);
516   TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,
      2);TrafficFromTo3(0, 14, 4, 4, 2);
517   TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,
      2);TrafficFromTo3(0, 14, 4, 4, 2);
518   TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,
      2);TrafficFromTo3(0, 14, 4, 4, 2);
519   TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,
      2);TrafficFromTo3(0, 14, 4, 4, 2);
520   TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,
      2);TrafficFromTo3(0, 14, 4, 4, 2);
521   TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,
```

```
        2);TrafficFromTo3(0, 14, 4, 4, 2);
522     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
523     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
524     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
525
526      TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,               ↵
         2);TrafficFromTo3(0, 14, 4, 4, 2);
527     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
528     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
529     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
530     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
531     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
532     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
533     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
534     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
535     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 14, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
536
537
538     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
539     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
540     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
541     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
542     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
543     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
544     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
545     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
546     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
547     TrafficFromTo3(0, 14, 4, 4, 2);TrafficFromTo3(0, 12, 4, 4,                ↵
        2);TrafficFromTo3(0, 14, 4, 4, 2);
548
549     TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
        3);TrafficFromTo3(0, 14, 5, 5, 3);
550     TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
        3);TrafficFromTo3(0, 14, 5, 5, 3);
551     TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
        3);TrafficFromTo3(0, 14, 5, 5, 3);
552     TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
        3);TrafficFromTo3(0, 14, 5, 5, 3);
553     TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
        3);TrafficFromTo3(0, 14, 5, 5, 3);
```

```
554         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
555         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
556         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
557         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
558         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
559
560         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
561         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
562         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
563         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
564         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
565         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
566         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
567         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
568         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
569         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
570
571         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
572         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
573         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
574         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
575         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
576         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
577         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
578         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
579         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
580         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
581
582         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
583         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
584         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
585         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,                ↵
            3);TrafficFromTo3(0, 14, 5, 5, 3);
```

```
586         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
587         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
588         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
589         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
590         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
591         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
592
593          TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
             3);TrafficFromTo3(0, 14, 5, 5, 3);
594         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
595         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
596         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
597         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
598         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
599         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
600         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
601         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
602         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
603
604         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
605         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
606         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
607         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
608         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
609         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
610         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
611         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
612         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
613         TrafficFromTo3(0, 14, 5, 5, 3);TrafficFromTo3(0, 14, 5, 5,
            3);TrafficFromTo3(0, 14, 5, 5, 3);
614
615
616     // Traffic 1
617         TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
            1);TrafficFromTo3(1, 14, 3, 3, 1);
618         TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
            1);TrafficFromTo3(1, 14, 3, 3, 1);
```

```
619       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
620       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
621       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
622       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
623      TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
         1);TrafficFromTo3(1, 14, 3, 3, 1);
624       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
625       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
626       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
627
628       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
629       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
630       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
631       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
632       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
633       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
634       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
635       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
636       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
637       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
638
639       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
640       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
641       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
642       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
643       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
644       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
645       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
646       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
647       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
648       TrafficFromTo3(1, 14, 3, 3, 1);TrafficFromTo3(1, 14, 3, 3,
          1);TrafficFromTo3(1, 14, 3, 3, 1);
649
650       TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
          2);TrafficFromTo3(1, 14, 4, 4, 2);
```

```
651     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
652     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
653     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
654     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
655     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
656     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
657     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
658     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
659     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
660
661     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
662     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
663     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
664     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
665     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
666     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
667     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
668     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
669     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
670     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
671
672     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
673     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
674     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
675     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
676     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
677     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
678     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
679     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
680     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
681     TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
        2);TrafficFromTo3(1, 14, 4, 4, 2);
682
```

```
683         TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
            2);TrafficFromTo3(1, 14, 4, 4, 2);
684         TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
            2);TrafficFromTo3(1, 14, 4, 4, 2);
685         TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
            2);TrafficFromTo3(1, 14, 4, 4, 2);
686         TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
            2);TrafficFromTo3(1, 14, 4, 4, 2);
687         TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
            2);TrafficFromTo3(1, 14, 4, 4, 2);
688         TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
            2);TrafficFromTo3(1, 14, 4, 4, 2);
689         TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
            2);TrafficFromTo3(1, 14, 4, 4, 2);
690         TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
            2);TrafficFromTo3(1, 14, 4, 4, 2);
691         TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
            2);TrafficFromTo3(1, 14, 4, 4, 2);
692         TrafficFromTo3(1, 14, 4, 4, 2);TrafficFromTo3(1, 14, 4, 4,
            2);TrafficFromTo3(1, 14, 4, 4, 2);
693
694         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
695         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
696         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
697         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
698         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
699         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
700         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
701         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
702         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
703         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
704
705         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
706         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
707         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
708         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
709         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
710         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
711         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
712         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
713         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
            3);TrafficFromTo3(1, 14, 5, 5, 3);
714         TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
```

```
        3);TrafficFromTo3(1, 14, 5, 5, 3);
715
716     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
717     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
718     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
719     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
720     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
721     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
722     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
723     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
724     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
725     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
726
727     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
728     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
729     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
730     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
731     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
732     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
733     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
734     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
735     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
736     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
737
738     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
739     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
740     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
741     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
742     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
743     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
744     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
745     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
        3);TrafficFromTo3(1, 14, 5, 5, 3);
746     TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,
```

```
      3);TrafficFromTo3(1, 14, 5, 5, 3);
747   TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,                    ↵
      3);TrafficFromTo3(1, 14, 5, 5, 3);
748
749   TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,                    ↵
      3);TrafficFromTo3(1, 14, 5, 5, 3);
750   TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,                    ↵
      3);TrafficFromTo3(1, 14, 5, 5, 3);
751   TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,                    ↵
      3);TrafficFromTo3(1, 14, 5, 5, 3);
752   TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,                    ↵
      3);TrafficFromTo3(1, 14, 5, 5, 3);
753   TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,                    ↵
      3);TrafficFromTo3(1, 14, 5, 5, 3);
754   TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,                    ↵
      3);TrafficFromTo3(1, 14, 5, 5, 3);
755   TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,                    ↵
      3);TrafficFromTo3(1, 14, 5, 5, 3);
756   TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,                    ↵
      3);TrafficFromTo3(1, 14, 5, 5, 3);
757   TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,                    ↵
      3);TrafficFromTo3(1, 14, 5, 5, 3);
758   TrafficFromTo3(1, 14, 5, 5, 3);TrafficFromTo3(1, 14, 5, 5,                    ↵
      3);TrafficFromTo3(1, 14, 5, 5, 3);
759
760
761
762     for (double i = 20000; i < Case1_conf::SIM_STOP_Sec * 1000; i += 1000) {
763         Simulator::Schedule(MilliSeconds(i), &Case3::ScheduleSchedulingRateUpdate,  ↵
          this);
764      }
765       for (double i = 10001; i < Case1_conf::SIM_STOP_Sec * 1000; i += 1000) {
766
767         Simulator::Schedule(MilliSeconds(i), &Case3::AverageScheduleRate, this);
768      }
769
770
771
772
773     Config::ConnectWithoutContext("/NodeList/*/$ns3::Ipv4L3Protocol/Rx",           ↵
      MakeCallback(&MacRecv));
774  }
775
776  void Case3::TrafficFromTo1(int from, int to, int ProtocolType, int TrafficType,  ↵
     int ToS)
777  {
778      uint16_t sinkPort = 8080;
779      Address                                                                     ↵
      sinkAddress(InetSocketAddress(nodes.Get(to)->GetObject<Ipv4>()->GetAddress(1,  ↵
      0).GetLocal(), sinkPort));
780      Ptr<Socket> TrafficSocket;
781      Ptr<TosApp> app1 = CreateObject<TosApp>(ToS, TrafficType);
782      if (ProtocolType == 3) { // edit 13/05   VoIP traffic
783          TrafficSocket = Socket::CreateSocket(nodes.Get(from),                   ↵
          UdpSocketFactory::GetTypeId());
784          app1->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,     ↵
          (80000/1600)*Case1_conf::SIM_STOP_Sec, DataRate("80000"),1,1,1);
785
786      } else if (ProtocolType == 4) { // edit 13/05   Video Conferencing traffic
787          TrafficSocket = Socket::CreateSocket(nodes.Get(from),                   ↵
          UdpSocketFactory::GetTypeId());
```

```
788         app1->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (384000/11200)*Case1_conf::SIM_STOP_Sec, DataRate("384000"),1,1,1);
789         // video conferencing rate 128-384 kbits/sec,
            https://en.wikipedia.org/wiki/Bit_rate
790     } else if (ProtocolType == 5) { // edit 13/05    (FTP) traffic  TcpSocketFactory
791         TrafficSocket =
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
792         app1->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (256000/2800)*Case1_conf::SIM_STOP_Sec, DataRate("256000"),1,1,1);
793
794     } else if (ProtocolType == 6) { // edit 13/05   // Database Query message
    traffic - Best effort traffic
795         TrafficSocket =
            Socket::CreateSocket(nodes.Get(from),UdpSocketFactory::GetTypeId());
796         app1->Setup(TrafficSocket, sinkAddress,
            Case1_conf::TRAFFICPKTSIZE,(81920/4096)*Case1_conf::SIM_STOP_Sec ,
            DataRate("81920"),1,1,1);
797
798     }
799     nodes.Get(from)->AddApplication(app1);
800
801     app1->SetStartTime(Seconds(1.));
802 }
803 void Case3::TrafficFromTo2(int from, int to, int ProtocolType, int TrafficType,
    int ToS)
804 {
805     uint16_t sinkPort = 8080;
806     Address
    sinkAddress(InetSocketAddress(nodes.Get(to)->GetObject<Ipv4>()->GetAddress(1,
    0).GetLocal(), sinkPort));
807     Ptr<Socket> TrafficSocket;
808     Ptr<TosApp> app2 = CreateObject<TosApp>(ToS, TrafficType);
809     if (ProtocolType == 3) { // edit 13/05   VoIP traffic
810         TrafficSocket = Socket::CreateSocket(nodes.Get(from),
            UdpSocketFactory::GetTypeId());
811         app2->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (80000/1600)*Case1_conf::SIM_STOP_Sec, DataRate("80000"),1,1,2);
812
813     } else if (ProtocolType == 4) { // edit 13/05   Video Conferencing traffic
814         TrafficSocket = Socket::CreateSocket(nodes.Get(from),
            UdpSocketFactory::GetTypeId());
815         app2->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (384000/11200)*Case1_conf::SIM_STOP_Sec, DataRate("384000"),1,1,2);
816         // video conferencing rate 128-384 kbits/sec,
            https://en.wikipedia.org/wiki/Bit_rate
817     } else if (ProtocolType == 5) { // edit 13/05    (FTP) traffic  TcpSocketFactory
818         TrafficSocket =
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
819         app2->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (256000/2800)*Case1_conf::SIM_STOP_Sec, DataRate("256000"),1,1,2);
820
821     } else if (ProtocolType == 6) { // edit 13/05   // Database Query message
    traffic - Best effort traffic
822         TrafficSocket =
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
823         app2->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (81920/4096)*Case1_conf::SIM_STOP_Sec, DataRate("81920"),1,1,2);
824     }
825     nodes.Get(from)->AddApplication(app2);
826
827     app2->SetStartTime(Seconds(1.));
```

```
828 }
829 void Case3::TrafficFromTo3(int from, int to, int ProtocolType, int TrafficType,
    int ToS)
830 {
831     uint16_t sinkPort = 8080;
832     Address
        sinkAddress(InetSocketAddress(nodes.Get(to)->GetObject<Ipv4>()->GetAddress(1,
        0).GetLocal(), sinkPort));
833     Ptr<Socket> TrafficSocket;
834     Ptr<TosApp> app3 = CreateObject<TosApp>(ToS, TrafficType);
835     if (ProtocolType == 3) { // edit 13/05   VoIP traffic
836         TrafficSocket = Socket::CreateSocket(nodes.Get(from),
            UdpSocketFactory::GetTypeId());
837         app3->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (80000/1600)*Case1_conf::SIM_STOP_Sec, DataRate("80000"),1,1,3);
838
839     } else if (ProtocolType == 4) { // edit 13/05   Video Conferencing traffic
840         TrafficSocket = Socket::CreateSocket(nodes.Get(from),
            UdpSocketFactory::GetTypeId());
841         app3->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (384000/11200)*Case1_conf::SIM_STOP_Sec, DataRate("384000"),1,1,3);
842         // video conferencing rate 128-384 kbits/sec,
            https://en.wikipedia.org/wiki/Bit_rate
843     } else if (ProtocolType == 5) { // edit 13/05    (FTP) traffic  TcpSocketFactory
844         TrafficSocket =
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
845         app3->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (256000/2800)*Case1_conf::SIM_STOP_Sec, DataRate("256000"),1,1,3);
846
847     }   else if (ProtocolType == 6) { // edit 13/05   // Database Query message
    traffic - Best effort traffic
848         TrafficSocket =
            Socket::CreateSocket(nodes.Get(from),TcpSocketFactory::GetTypeId());
849         app3->Setup(TrafficSocket, sinkAddress, Case1_conf::TRAFFICPKTSIZE,
            (81920/4096)*Case1_conf::SIM_STOP_Sec, DataRate("81920"),1,1,3);
850
851     }
852     nodes.Get(from)->AddApplication(app3);
853
854     app3->SetStartTime(Seconds(1.));
855
856
857 }
858
859 void Case3::ScheduleSchedulingRateUpdate(void)
860 {
861     double Length810[4], Length811[4], Length812[4];
862     double Length910[4], Length911[4], Length912[4];
863     double MaxQueueLength1 [4], MaxQueueLength2 [4], MaxQueueLength [4];
864     for (int i = 1; i <= 3; i++) {
865     Length810[i] = queue3810->GetAverageQueueLength1(i);
866     Length811[i] = queue3811->GetAverageQueueLength1(i);
867     Length812[i] = queue3812->GetAverageQueueLength1(i);
868     Length910[i] = queue3910->GetAverageQueueLength1(i);
869     Length911[i] = queue3911->GetAverageQueueLength1(i);
870     Length912[i] = queue3912->GetAverageQueueLength1(i);
871
872     MaxQueueLength1 [i] = MaximumAverageQueueLength(Length810[i], Length811[i],
    Length812[i]);
873     MaxQueueLength2 [i] = MaximumAverageQueueLength(Length910[i], Length911[i],
    Length912[i]);
```

```
874     MaxQueueLength [i] = MaximumAverageQueueLength(0, MaxQueueLength1 [i],
        MaxQueueLength2 [i]);
875      }
876        // either section A or B work
877        //Section A
878        queue348->SetSchedulingRateByFactor(1, MaxQueueLength[1] /
           (float)Case1_conf::SC1BYTEBUFFER);
879        queue379->SetSchedulingRateByFactor(1, MaxQueueLength[1] /
           (float)Case1_conf::SC1BYTEBUFFER);
880
881        queue348->SetSchedulingRateByFactor(2, MaxQueueLength[2] /
           (float)Case1_conf::SC2BYTEBUFFER);
882        queue379->SetSchedulingRateByFactor(2, MaxQueueLength[2] /
           (float)Case1_conf::SC2BYTEBUFFER);
883
884        queue348->SetSchedulingRateByFactor(3, MaxQueueLength[3] /
           (float)Case1_conf::SC3BYTEBUFFER);
885        queue379->SetSchedulingRateByFactor(3, MaxQueueLength[3] /
           (float)Case1_conf::SC3BYTEBUFFER);
886
887
888      //Section B
889
890          queue348->SetSchedulingRateByLinearFactor(1, MaxQueueLength[1]);
891          queue379->SetSchedulingRateByLinearFactor(1, MaxQueueLength[1]);
892
893          queue348->SetSchedulingRateByLinearFactor(2, MaxQueueLength[2]);
894          queue379->SetSchedulingRateByLinearFactor(2, MaxQueueLength[2]);
895
896          queue348->SetSchedulingRateByLinearFactor(3, MaxQueueLength[3]);
897          queue379->SetSchedulingRateByLinearFactor(3, MaxQueueLength[3]);
898
899  }
900
901
902  void Case3::AverageScheduleRate(void)
903  {
904      double length810[4], length811[4], length812[4];
905      double length910[4], length911[4], length912[4];
906      double delay810[4], delay811[4], delay812[4];
907      double delay910[4], delay911[4], delay912[4];
908      double Minqueuedelay1 [4], Minqueuedelay2 [4], Minqueuedelay [4];
909      double Maxqueuelength1 [4], Maxqueuelength2 [4], Maxqueuelength [4];
910      for (int i = 1; i <= 3; i++) {
911      delay810[i] = queue3810->GetAverageQueueDelay(i);
912      delay811[i] = queue3811->GetAverageQueueDelay(i);
913      delay812[i] = queue3812->GetAverageQueueDelay(i);
914      delay910[i] = queue3910->GetAverageQueueDelay(i);
915      delay911[i] = queue3911->GetAverageQueueDelay(i);
916      delay912[i] = queue3912->GetAverageQueueDelay(i);
917      Minqueuedelay1 [i] = MinimumAverageQueueDelay
         (delay810[i],delay811[i],delay812[i]);
918      Minqueuedelay2 [i] = MinimumAverageQueueDelay
         (delay910[i],delay911[i],delay912[i]);
919      Minqueuedelay  [i] = MinimumAverageQueueDelay (0,Minqueuedelay1
         [i],Minqueuedelay2 [i]);
920      std::cout<<Simulator::Now() <<"  "<<"delay810:"<<" "<<delay810[i]<<"
         "<<"delay811:"<<" "<<delay811[i]<<"  "<<"delay812:"<<" "<<delay812[i]<<"
         "<<i<<" "<<"minimum average delay1:"<<"  "<<Minqueuedelay1 [i]<<std::endl;
921      std::cout<<Simulator::Now() <<"  "<<"delay910:"<<" "<<delay910[i]<<"
         "<<"delay911:"<<" "<<delay911[i]<<"  "<<"delay912:"<<" "<<delay912[i]<<"
```

```
      "<<i<<" "<<"minimum average delay2:"<<"  "<<Minqueuedelay2 [i]<<std::endl;
922   std::cout<<Simulator::Now() <<"  "<<"Minqueuesdelay:"<<" "<<Minqueuedelay [i]<<std::endl;
923
924
925   length810[i] = queue3810->GetAverageQueueLength2(i);
926   length811[i] = queue3811->GetAverageQueueLength2(i);
927   length812[i] = queue3812->GetAverageQueueLength2(i);
928   length910[i] = queue3910->GetAverageQueueLength2(i);
929   length911[i] = queue3911->GetAverageQueueLength2(i);
930   length912[i] = queue3912->GetAverageQueueLength2(i);
931   Maxqueuelength1 [i] = MaximumAverageQueueLength(length810[i], length811[i], length812[i]);
932   Maxqueuelength2 [i] = MaximumAverageQueueLength(length910[i], length911[i], length912[i]);
933   Maxqueuelength [i] = MaximumAverageQueueLength(0, Maxqueuelength1 [i], Maxqueuelength2 [i]);
934   std::cout<<Simulator::Now()<<"length810:"<<" "<<length810[i]<<" "<<"length811:"<<" "<<length811[i]<<"  "<<"length812:"<<" "<<length812[i]<<" "<<i<<" "<<"maximum average length:"<<"  "<<Maxqueuelength1 [i]<<std::endl;
935   std::cout<<Simulator::Now()<<"length910:"<<" "<<length910[i]<<" "<<"length911:"<<" "<<length911[i]<<"  "<<"length912:"<<" "<<length912[i]<<" "<<i<<" "<<"maximum average length:"<<"  "<<Maxqueuelength2 [i]<<std::endl;
936   std::cout<<Simulator::Now()<<"Maximumqueuelengths:"<<" "<<Maxqueuelength [i]<<std::endl;
937
938   queue3810->SetSchedulingRateByAverage(i, Maxqueuelength [1], Maxqueuelength [2], Maxqueuelength [3], Minqueuedelay[1], Minqueuedelay[2], Minqueuedelay[3]);
939   queue3811->SetSchedulingRateByAverage(i, Maxqueuelength [1], Maxqueuelength [2], Maxqueuelength [3], Minqueuedelay[1], Minqueuedelay[2], Minqueuedelay[3]);
940   queue3812->SetSchedulingRateByAverage(i, Maxqueuelength [1], Maxqueuelength [2], Maxqueuelength [3], Minqueuedelay[1], Minqueuedelay[2], Minqueuedelay[3]);
941   queue3910->SetSchedulingRateByAverage(i, Maxqueuelength [1], Maxqueuelength [2], Maxqueuelength [3], Minqueuedelay[1], Minqueuedelay[2], Minqueuedelay[3]);
942   queue3911->SetSchedulingRateByAverage(i, Maxqueuelength [1], Maxqueuelength [2], Maxqueuelength [3], Minqueuedelay[1], Minqueuedelay[2], Minqueuedelay[3]);
943   queue3912->SetSchedulingRateByAverage(i, Maxqueuelength [1], Maxqueuelength [2], Maxqueuelength [3], Minqueuedelay[1], Minqueuedelay[2], Minqueuedelay[3]);
944
945
946
947     }
948
949  }
950  double Case3::MinimumAverageQueueDelay(double x, double y, double z)
951 {    double minqueuedelay;
952     if (x !=0 && y !=0 && z!=0)
953         {
954          minqueuedelay = x;
955         if (y< minqueuedelay)
956         {
957         minqueuedelay = y;
958         }
959         if (z < minqueuedelay)
960         {
961         minqueuedelay = z;
962         }
963         }
964         else if (x ==0 && y !=0 && z !=0)
965         {
966          minqueuedelay = y;
```

```
          if (z < minqueuedelay)
          {
          minqueuedelay = z;
          }
         }

         else if (x !=0 && y ==0 && z !=0)
         {
          minqueuedelay = x;
          if (z < minqueuedelay)
          {
          minqueuedelay = z;
          }
         }

         else if (x !=0 && y !=0 && z ==0)
         {
          minqueuedelay = x;
          if (y < minqueuedelay)
          {
          minqueuedelay = y;
          }
         }

         else if (x ==0 && y ==0 && z !=0)
         {
          minqueuedelay = z;
         }

         else if (x ==0 && y !=0 && z ==0)
         {
          minqueuedelay = y;
         }

         else if (x !=0 && y ==0 && z ==0)
         {
          minqueuedelay = x;
         }
         else if (x ==0 && y ==0 && z ==0)
         {
          minqueuedelay = 0.1;
         }
         return minqueuedelay;
}
double Case3::MaximumAverageQueueLength(double x, double y, double z)
{
    double maxqueuelebgth = x;
       if (y>maxqueuelebgth)
       {
       maxqueuelebgth = y;
       }
       if (z > maxqueuelebgth)
       {
        maxqueuelebgth = z;
       }
        return maxqueuelebgth;
}
Case3::~Case3()
{
    // dtor
}
```

A-6: Simulation Parameters Definition Code

```
#ifndef CASE1_CONF_H
#define CASE1_CONF_H

#include "ns3/core-module.h"


using namespace ns3;

class Case1_conf
{
public:

    static  double   CONSTANTRADVAR ;
    static  double   EXPONRADVARMEAN ;
    static  double   EXPONRADVARBOUND  ;
    static  double   PARETORADVARMEAN  ;
    static  double   PARETORADVARSAHPE  ;

// update for on-off application
    static  double   ONTIMEEXPONMEAN ;
    static  double   ONTIMEEXPONBOUND  ;
    static  double   OFFTIMEEXPONMEAN ;
    static  double   OFFTIMEEXPONBOUND  ;

  ////////////////////////////
    static const double   EXPONPKTSIZEMEAN  ;
    static const double   EXPONPKTSIZEBOUND  ;

    static const int   REDSC1THIGH;
    static const int   REDSC2THIGH;
    static const int   REDSC3THIGH;

    static const double   REDSC1HDROP;
    static const double   REDSC2HDROP;
    static const double   REDSC3HDROP;


    static const int   REDSC1TLOW;
    static const int   REDSC2TLOW;
    static const int   REDSC3TLOW;

    static const double   REDSC1LDROP;
    static const double   REDSC2LDROP;
    static const double   REDSC3LDROP;


    static  int   SC1PKTBUFFER;
    static  int   SC2PKTBUFFER;
    static  int   SC3PKTBUFFER;

    static  int   SC1BYTEBUFFER;
    static  int   SC2BYTEBUFFER;
    static  int   SC3BYTEBUFFER;

    static float SC1CONGESTIONFATORALPHAL;
    static float SC2CONGESTIONFATORALPHAL;
    static float SC3CONGESTIONFATORALPHAL;

    static float SC1CONGESTIONFATORALPHAH;
    static float SC2CONGESTIONFATORALPHAH;
    static float SC3CONGESTIONFATORALPHAH;
```

```
62
63      static float CONGESTIONFATBELTA1;
64      static float CONGESTIONFATBELTA2;
65      static float CONGESTIONFATBELTA3;
66
67      static float SC1BETAMAX;
68      static float SC2BETAMAX;
69      static float SC3BETAMAX;
70
71      static float SC1INITWEIGHT;
72      static float SC2INITWEIGHT;
73      static float SC3INITWEIGHT;
74
75      static uint64_t LINKCAPACITY;
76      static uint64_t LINKDELAY;
77      static float INTERFRAMEGAP;
78
79   static uint64_t DomainsLINKCAPACITY;
80   static uint64_t DestinationLINKCAPACITY;
81   static uint64_t UpstreamLINKCAPACITY;
82   static uint64_t DownstreamLINKCAPACITY;
83
84      static uint64_t TRAFFICDATARATE;
85      static uint32_t TRAFFICPKTSIZE;
86      static uint64_t TRAFFICPKTNUM;
87
88      static double SIM_STOP_Sec;
89
90      static double ROWFORAVGQLENGTH;
91
92
93
94  };
95
96
97
98
99  #endif // CASE1_CONF_H
100
```

```cpp
#include "Case1_conf.h"
using namespace ns3;



 double  Case1_conf::CONSTANTRADVAR =0.1;
 double  Case1_conf::EXPONRADVARMEAN = 3.14;
 double  Case1_conf::EXPONRADVARBOUND = 0.0;
 double  Case1_conf::PARETORADVARMEAN = 3.0;
 double  Case1_conf::PARETORADVARSAHPE = 2.0;

// update for on-off application
double Case1_conf::ONTIMEEXPONMEAN = 3;
double Case1_conf::ONTIMEEXPONBOUND = 0;
double Case1_conf::OFFTIMEEXPONMEAN = 3;
double Case1_conf::OFFTIMEEXPONBOUND = 0;

const double  Case1_conf::EXPONPKTSIZEMEAN = 0.5;
const double  Case1_conf::EXPONPKTSIZEBOUND = 0.0;


const int   Case1_conf::REDSC1THIGH=12800; //0.5* buffer size
const int   Case1_conf::REDSC2THIGH=89600;  //0.5* buffer size
const int   Case1_conf::REDSC3THIGH=22400; //0.5* buffer size


const double   Case1_conf::REDSC1HDROP=0.10;
const double   Case1_conf::REDSC2HDROP=0.10;
const double   Case1_conf::REDSC3HDROP=0.10;



const int   Case1_conf::REDSC1TLOW=12000;
const int   Case1_conf::REDSC2TLOW=78400;
const int   Case1_conf::REDSC3TLOW=18200;


const double   Case1_conf::REDSC1LDROP=0.10;
const double   Case1_conf::REDSC2LDROP=0.10;
const double   Case1_conf::REDSC3LDROP=0.10;


int   Case1_conf::SC1PKTBUFFER=2048;
int   Case1_conf::SC2PKTBUFFER=2048;
int   Case1_conf::SC3PKTBUFFER=2048;


int   Case1_conf::SC1BYTEBUFFER=25600;    //128*200
int   Case1_conf::SC2BYTEBUFFER=179200;   // 128*1400
int   Case1_conf::SC3BYTEBUFFER=44800; //128*350


float Case1_conf::SC1CONGESTIONFATORALPHAH=0.9;
float Case1_conf::SC2CONGESTIONFATORALPHAH=0.9;
float Case1_conf::SC3CONGESTIONFATORALPHAH=0.9;

float Case1_conf::SC1CONGESTIONFATORALPHAL=0.46875;
float Case1_conf::SC2CONGESTIONFATORALPHAL=0.4375;
float Case1_conf::SC3CONGESTIONFATORALPHAL=0.40625;


```

```
62  float Case1_conf::CONGESTIONFATBELTA1=0.3;
63  float Case1_conf::CONGESTIONFATBELTA2=0.6;
64  float Case1_conf::CONGESTIONFATBELTA3=0.9;
65
66  float Case1_conf::SC1BETAMAX=0.9;
67  float Case1_conf::SC2BETAMAX=0.9;
68  float Case1_conf::SC3BETAMAX=0.9;
69
70  float Case1_conf::SC1INITWEIGHT=1;
71  float Case1_conf::SC2INITWEIGHT=2;
72  float Case1_conf::SC3INITWEIGHT=4;
73
74  uint64_t Case1_conf::LINKCAPACITY=2000000;
75  uint64_t Case1_conf::LINKDELAY=1;
76  float Case1_conf::INTERFRAMEGAP=2;
77
78  uint64_t Case1_conf::UpstreamLINKCAPACITY=2000000;
79  uint64_t Case1_conf::DownstreamLINKCAPACITY=2000000;
80  uint64_t Case1_conf::DomainsLINKCAPACITY=4000000;
81  uint64_t Case1_conf::DestinationLINKCAPACITY=4000000;
82
83
84  uint64_t Case1_conf::TRAFFICDATARATE=1000000000;
85  uint32_t Case1_conf::TRAFFICPKTSIZE=800;
86  uint64_t Case1_conf::TRAFFICPKTNUM=10000000;
87
88  double Case1_conf::SIM_STOP_Sec=200;
89
90  double Case1_conf::ROWFORAVGQLENGTH=0.1;
91
```

A-7: End to End Average Delay Analysis Code

```
#include <iostream>
#include <fstream>
#include <string>
#include <stdint.h>
#include <sstream>
#include <cmath>
#include <map>
#include <iterator>
#include <vector>
using namespace std;


int main()
{
//Use g++ -std=c++14 -Wall -Wextra -pedantic-errors -O3 -o ETE ETE.cpp
//O3 - optimise
//std=c++14 - the current standard for C++
//pedantic-errors - the International Standard, not your home grown GNU dialect
// to run the program ./ETE


 std::map<int,double_t> Pkt_arrivalmap; // identify the map that contain the arrival information of packets.
 std::map<int,double_t> SC1Pkt_arrivalmap;
 std::map<int,double_t> SC2Pkt_arrivalmap;
 std::map<int,double_t> SC3Pkt_arrivalmap;
 std::map<int,double_t> SC4Pkt_arrivalmap;

 std::map<int,double_t> Pkt_creatingmap; // identify the map that contain the creating information of packets.
 std::map<int,double_t> SC1Pkt_creatingmap;
 std::map<int,double_t> SC2Pkt_creatingmap;
 std::map<int,double_t> SC3Pkt_creatingmap;
 std::map<int,double_t> SC4Pkt_creatingmap;

 std::vector<double_t> TotalTrafficETED;
 std::vector<double_t> SC1TrafficETED;
 std::vector<double_t> SC2TrafficETED;
 std::vector<double_t> SC3TrafficETED;
 std::vector<double_t> SC4TrafficETED;

double_t ETEDforeachclass(int i);

int pos,pos2,pos3;
double_t TTETED=0;
double_t SC1TETED=0;
double_t SC2TETED=0;
double_t SC3TETED=0;
double_t SC4TETED=0;

string line,search,word1="ns",word2="Pkt_create", word3="Pkt_recv";  // create an object from string use to read each line in file.
ifstream inf; // create an object from class of input file stream for reading only

//// to store columns values of 123.txt file (receiving) into variables, then store variables in container arrival map:
// search each line of file 123.txt on Pkt_create and print this line   (okay)

inf.open ("123.txt");  // open file “123.txt” for reading
if (inf.is_open())  // condition to check if it is possible to open file or not. (if it exist), then open it and do following prcess.
```

```
{

    while (getline(inf,line)) // loop continue until reaching the end of file.
    {

        if (line.find("Pkt_recv") != string::npos)   // search on "Pkt_recv" each line
        {
        pos=line.find(word1); // take the length of "ns" in string line.
        pos3=line.find(word3);// take the length of "Pkt_recv" in string line.
        line.replace(pos,word1.length(),"  "); // to remove "ns"
        line.replace(pos3,word3.length(),"  "); // to remove "Pkt_recv"
        istringstream iss (line);
        istringstream iss1 (line);
        double_t column1;
        double_t column3;
        double_t column2;
        iss >> column1 >> column2;
        //cout << column1 <<"  "<< column2 <<"  "<< " "<< endl;
        Pkt_arrivalmap[column1]=column2;
        ////////////////////////////
        iss1 >> column1 >> column2 >> column3;
        //cout << column1 <<"  "<< column2 <<"  "<< column3 << " "<< endl;
        if (column3==1)
        {
                SC1Pkt_arrivalmap[column1]=column2;
        }
                else if (column3==2)
        {
                SC2Pkt_arrivalmap[column1]=column2;
        }
                else if (column3==3)
        {
                SC3Pkt_arrivalmap[column1]=column2;
        }

                else

                SC4Pkt_arrivalmap[column1]=column2;

        }



}


inf.close();
}

else cout << "Unable to open file inf \n";

//// to store columns values of 123.txt file (creating) into variables, then store variables in container creatingmap:
// search each line of file 123.txt on Pkt_create and print this line   (okay)

inf.open ("123.txt");  // open file “123.txt” for reading
if (inf.is_open())  // condition to check if it is possible to open file or not. (if it exist), then open it and do following prcess.
{

```

```
115     while (getline(inf,line)) // loop continue until reaching the end of file.
116     {
117
118         if (line.find("Pkt_create") != string::npos)   // search on "Pkt_create"
        each line
119         {
120         pos=line.find(word1); // take the length of "ns" in string line.
121         pos2=line.find(word2);// take the length of "Pkt_create" in string line.
122         line.replace(pos,word1.length(),"  "); // to remove "ns"
123         line.replace(pos2,word2.length(),"  "); // to remove "Pkt_create"
124         istringstream iss (line);
125         istringstream iss1 (line);
126         double_t column1;
127         double_t column3;
128         double_t column2;
129         iss >> column1 >> column2;
130         //cout << column1 <<"  "<< column2 <<"  "<< " "<< endl;
131         Pkt_creatingmap[column1]=column2;
132         iss1 >> column1 >> column2 >> column3;
133         //cout << column1 <<"  "<< column2 <<"  "<< column3 << " "<< endl;
134         if (column3==1)
135         {
136                 SC1Pkt_creatingmap[column1]=column2;
137         }
138         else if (column3==2)
139         {
140                 SC2Pkt_creatingmap[column1]=column2;
141         }
142         else if (column3==3)
143         {
144                 SC3Pkt_creatingmap[column1]=column2;
145         }
146
147         else
148
149                 SC4Pkt_creatingmap[column1]=column2;
150
151
152         }
153 }
154
155 inf.close();
156 }
157 else cout << "Unable to open file inf \n";
158
159
160 ////////////////////////////////////// For total service classes traffic
161 auto Pkt_arrivalmap_it = Pkt_arrivalmap.begin();
162 auto Pkt_arrivalmap_end = Pkt_arrivalmap.end();
163 const int lowest_key=Pkt_arrivalmap_it->first;
164 auto Pkt_creatingmap_it = Pkt_creatingmap.lower_bound(lowest_key);
165 auto Pkt_creatingmap_end = Pkt_creatingmap.end();
166
167 while (Pkt_arrivalmap_it != Pkt_arrivalmap_end && Pkt_creatingmap_it !=
    Pkt_creatingmap_end)
168 {
169     if (Pkt_arrivalmap_it ->first < Pkt_creatingmap_it ->first)
170       ++ Pkt_arrivalmap_it;
171     else
172        if (Pkt_arrivalmap_it ->first == Pkt_creatingmap_it ->first)
173        {
```

```
174         TotalTrafficETED.push_back(Pkt_arrivalmap_it ->second - Pkt_creatingmap_it
            ->second);
175         Pkt_arrivalmap_it=Pkt_arrivalmap.erase(Pkt_arrivalmap_it);
176         Pkt_creatingmap_it = Pkt_creatingmap.erase(Pkt_creatingmap_it);
177         }
178         else ++Pkt_creatingmap_it;
179
180 }
181
182
183 for (std::vector<double_t>::iterator it = TotalTrafficETED.begin(); it !=
    TotalTrafficETED.end(); it++)
184 {
185     TTETED+=(*it/TotalTrafficETED.size())/1000000000;
186 }
187     cout << "Total Traffic ETED" << "  "<< TTETED <<"  "<<"Sec."<< endl;
188
189 ////////////////////////////////////// For SC1 traffic
190
191 auto SC1Pkt_arrivalmap_it = SC1Pkt_arrivalmap.begin();
192 auto SC1Pkt_arrivalmap_end = SC1Pkt_arrivalmap.end();
193 const int SC1lowest_key=SC1Pkt_arrivalmap_it->first;
194 auto SC1Pkt_creatingmap_it = SC1Pkt_creatingmap.lower_bound(SC1lowest_key);
195 auto SC1Pkt_creatingmap_end = SC1Pkt_creatingmap.end();
196
197 while (SC1Pkt_arrivalmap_it != SC1Pkt_arrivalmap_end && SC1Pkt_creatingmap_it !=
    SC1Pkt_creatingmap_end)
198 {
199     if (SC1Pkt_arrivalmap_it ->first < SC1Pkt_creatingmap_it ->first)
200       ++ SC1Pkt_arrivalmap_it;
201     else
202        if (SC1Pkt_arrivalmap_it ->first == SC1Pkt_creatingmap_it ->first)
203        {
204        SC1TrafficETED.push_back(SC1Pkt_arrivalmap_it ->second -
           SC1Pkt_creatingmap_it ->second);
205        SC1Pkt_arrivalmap_it=SC1Pkt_arrivalmap.erase(SC1Pkt_arrivalmap_it);
206        SC1Pkt_creatingmap_it = SC1Pkt_creatingmap.erase(SC1Pkt_creatingmap_it);
207        }
208        else ++SC1Pkt_creatingmap_it;
209
210 }
211
212 for (std::vector<double_t>::iterator it = SC1TrafficETED.begin(); it !=
    SC1TrafficETED.end(); it++)
213 {
214     SC1TETED+=(*it/SC1TrafficETED.size())/1000000000;
215 }
216     cout << "SC1 Traffic ETED" << "  "<< SC1TETED <<"  "<<"Sec."<< endl;
217
218
219
220 ////////////////////////////////////// For SC2 traffic
221
222 auto SC2Pkt_arrivalmap_it = SC2Pkt_arrivalmap.begin();
223 auto SC2Pkt_arrivalmap_end = SC2Pkt_arrivalmap.end();
224 const int SC2lowest_key=SC2Pkt_arrivalmap_it->first;
225 auto SC2Pkt_creatingmap_it = SC2Pkt_creatingmap.lower_bound(SC2lowest_key);
226 auto SC2Pkt_creatingmap_end = SC2Pkt_creatingmap.end();
227
228 while (SC2Pkt_arrivalmap_it != SC2Pkt_arrivalmap_end && SC2Pkt_creatingmap_it !=
    SC2Pkt_creatingmap_end)
```

```
{
    if (SC2Pkt_arrivalmap_it ->first < SC2Pkt_creatingmap_it ->first)
      ++ SC2Pkt_arrivalmap_it;
    else
       if (SC2Pkt_arrivalmap_it ->first == SC2Pkt_creatingmap_it ->first)
       {
       SC2TrafficETED.push_back(SC2Pkt_arrivalmap_it ->second -
       SC2Pkt_creatingmap_it ->second);
       SC2Pkt_arrivalmap_it=SC2Pkt_arrivalmap.erase(SC2Pkt_arrivalmap_it);
       SC2Pkt_creatingmap_it = SC2Pkt_creatingmap.erase(SC2Pkt_creatingmap_it);
       }
       else ++SC2Pkt_creatingmap_it;

}

for (std::vector<double_t>::iterator it = SC2TrafficETED.begin(); it !=
SC2TrafficETED.end(); it++)
{
    SC2TETED+=(*it/SC2TrafficETED.size())/1000000000;
}
    cout << "SC2 Traffic ETED" << "  "<< SC2TETED <<"  "<<"Sec."<< endl;

////////////////////////////////////// For SC3 traffic

auto SC3Pkt_arrivalmap_it = SC3Pkt_arrivalmap.begin();
auto SC3Pkt_arrivalmap_end = SC3Pkt_arrivalmap.end();
const int SC3lowest_key=SC3Pkt_arrivalmap_it->first;
auto SC3Pkt_creatingmap_it = SC3Pkt_creatingmap.lower_bound(SC3lowest_key);
auto SC3Pkt_creatingmap_end = SC3Pkt_creatingmap.end();

while (SC3Pkt_arrivalmap_it != SC3Pkt_arrivalmap_end && SC3Pkt_creatingmap_it !=
SC3Pkt_creatingmap_end)
{
    if (SC3Pkt_arrivalmap_it ->first < SC3Pkt_creatingmap_it ->first)
      ++ SC3Pkt_arrivalmap_it;
    else
       if (SC3Pkt_arrivalmap_it ->first == SC3Pkt_creatingmap_it ->first)
       {
       SC3TrafficETED.push_back(SC3Pkt_arrivalmap_it ->second -
       SC3Pkt_creatingmap_it ->second);
       SC3Pkt_arrivalmap_it=SC3Pkt_arrivalmap.erase(SC3Pkt_arrivalmap_it);
       SC3Pkt_creatingmap_it = SC3Pkt_creatingmap.erase(SC3Pkt_creatingmap_it);
       }
       else ++SC3Pkt_creatingmap_it;

}

for (std::vector<double_t>::iterator it = SC3TrafficETED.begin(); it !=
SC3TrafficETED.end(); it++)
{
    SC3TETED+=(*it/SC3TrafficETED.size())/1000000000;
}
    cout << "SC3 Traffic ETED" << "  "<< SC3TETED <<"  "<<"Sec."<< endl;

////////////////////////////////////// For SC4 traffic

auto SC4Pkt_arrivalmap_it = SC4Pkt_arrivalmap.begin();
auto SC4Pkt_arrivalmap_end = SC4Pkt_arrivalmap.end();
const int SC4lowest_key=SC4Pkt_arrivalmap_it->first;
auto SC4Pkt_creatingmap_it = SC4Pkt_creatingmap.lower_bound(SC4lowest_key);
auto SC4Pkt_creatingmap_end = SC4Pkt_creatingmap.end();
```

```
285
286 while (SC4Pkt_arrivalmap_it != SC4Pkt_arrivalmap_end && SC4Pkt_creatingmap_it !=
    SC4Pkt_creatingmap_end)
287 {
288     if (SC4Pkt_arrivalmap_it ->first < SC4Pkt_creatingmap_it ->first)
289       ++ SC4Pkt_arrivalmap_it;
290     else
291        if (SC4Pkt_arrivalmap_it ->first == SC4Pkt_creatingmap_it ->first)
292        {
293        SC4TrafficETED.push_back(SC4Pkt_arrivalmap_it ->second -
           SC4Pkt_creatingmap_it ->second);
294        SC4Pkt_arrivalmap_it=SC4Pkt_arrivalmap.erase(SC4Pkt_arrivalmap_it);
295        SC4Pkt_creatingmap_it = SC4Pkt_creatingmap.erase(SC4Pkt_creatingmap_it);
296        }
297        else ++SC4Pkt_creatingmap_it;
298
299 }
300
301 for (std::vector<double_t>::iterator it = SC4TrafficETED.begin(); it !=
    SC4TrafficETED.end(); it++)
302 {
303     SC4TETED+=(*it/SC4TrafficETED.size())/1000000000;
304 }
305     cout << "SC4 Traffic ETED" << "  "<< SC4TETED <<"  "<<"Sec."<< endl;
306
307
308 cout << "======================================="<< endl;
309 cout << "Total number of arrived packets" << "  "<< TotalTrafficETED.size() <<"
    "<<"Packets"<< endl;
310 cout << "Total number of SC1 arrived packets" << "  "<< SC1TrafficETED.size() <<"
    " <<"Packets"<< endl;
311 cout << "Total number of SC2 arrived packets" << "  "<< SC2TrafficETED.size() <<"
    " <<"Packets"<< endl;
312 cout << "Total number of SC3 arrived packets" << "  "<< SC3TrafficETED.size() <<"
    " <<"Packets"<< endl;
313 cout << "Total number of SC4 arrived packets" << "  "<< SC4TrafficETED.size() <<"
    " <<"Packets"<< endl;
314
315
316
317
318
319 return 0;
320
321
322
323 }
324
325
```

A-8: Output file Analysis Code

```
//run the simulation
NS_LOG="RDWQueue" ./waf --run scratch/Case1-sim > 123.txt 2>&1
============================================================
//display the result
less 123.txt
============================================================
//number of packets from each source
for ((i=1;i<5;i++)); do echo "SC"$i" "$(cat 123.txt | grep create | awk -v b=$i '{if($4==b){print
$0}}' | wc -l); done
or
less 123.txt | grep "pkt size" | awk '{print $1}' | sort | uniq -c  .
or
less 123.txt | grep "pkt size" | awk '{print $6}' | sort | uniq -c
============================================================
//total number of packets
less 123.txt | grep "pkt size" | wc -l
============================================================
//schedule rate
To get the average queue scheduling rate within an upadate interval
less 123.txt | grep "Queue Scheduling Rate Report 'Queue ID'" | awk '{print $1"  "$6"  "$7"  "$8"
"$9"  "$10"  "$11"  "$12"  "$13}'| less

or

to display the scheduling rate for service class queues and current usage, the first three values
represent the scheduling rates and the last three values represent the current usage.
less 123.txt | grep "CalculateSchedulingRate 'Queue ID'" | less
============================================================
//congestion level alpha (DWFQ algorithm)
less 123.txt  | grep "ByFactor" | less

To display the congestion level in SC1
less 123.txt | grep "ByFactor case" | grep "'Queue ID' 1" | less

less 123.txt | grep "ByFactor case 1 'Queue ID' 1"| less
The first number represent the case, the second represent the queue ID that reduce its rate and the
third number represent the service class.

To display the congestion level in SC2
less 123.txt | grep "ByFactor case" | grep "'Queue ID' 2" | less
The first number represent the case, the second represent the queue ID that reduce its rate and the
third number represent the service class.

To display the congestion level in SC3
less 123.txt | grep "ByFactor case" | grep "'Queue ID' 3" | less

less 123.txt | grep "ByFactor case 1 'Queue ID' 3"| less
The first number represent the case, the second represent the queue ID that reduce its rate and the
third number represent the service class.

or

To display SC1 congestion level only (in case of using DWFQ algorithm):
less 123.txt | grep "ByFactor case 1 'Queue ID' 1"| less
less 123.txt | grep "ByFactor case 2 'Queue ID' 1"| less
less 123.txt | grep "ByFactor case 3 'Queue ID' 1"| less

To display SC2 congestion and cases of beta:
less 123.txt | grep "ByFactor case 1 'Queue ID' 2"| less
less 123.txt | grep "ByFactor case 2 'Queue ID' 2"| less
less 123.txt | grep "ByFactor case 3 'Queue ID' 2"| less

To display SC3 congestion and cases of beta:
less 123.txt | grep "ByFactor case 1 'Queue ID' 3"| less
less 123.txt | grep "ByFactor case 2 'Queue ID' 3"| less
less 123.txt | grep "ByFactor case 3 'Queue ID' 3"| less

============================================================
for any scenairo, the congestion level and beta values (in case of using DRAM-NFV)
less 123.txt | grep "ByLinearFactor case 3 'Queue ID' 1" | less
less 123.txt | grep "ByLinearFactor case 3 'Queue ID' 2" | less
less 123.txt | grep "ByLinearFactor case 3 'Queue ID' 3" | less

less 123.txt | grep "ByLinearFactor case 2 'Queue ID' 1" | less
```

```
less 123.txt | grep "ByLinearFactor case 2 'Queue ID' 2" | less
less 123.txt | grep "ByLinearFactor case 2 'Queue ID' 3" | less

less 123.txt | grep "ByLinearFactor case 1 'Queue ID' 1" | less
less 123.txt | grep "ByLinearFactor case 1 'Queue ID' 2" | less
less 123.txt | grep "ByLinearFactor case 1 'Queue ID' 3" | less
============================================================

//Display scheduling rates, GCD and bucket values for a queue

To display the scheduling rate only at specific queue
less 123.txt | grep "NEW SCHEDULE RATE VALUE 'Queue ID'" | less
or
less 123.txt | grep "CalculateSchedulingRate 'Queue ID'" | less
The first three values represent the scheduling rates

To display the GCD and buckets values at specific queue
less 123.txt | grep "NEW SCHEDULE RATE BUCKETS 'Queue ID'" | less

============================================================
// Display normalised values only for a queue

To display the service classes normalised values for specific queue
less 123.txt | grep "NEW SCHEDULE RATE NORMALIZED VALUE 'Queue ID'" | grep -v 'nan' |less
============================================================
Delay of all service classes at a queue

To get the average queue delay in MilliSecond within an upadate interval k
less 123.txt | grep "Average Queue Delay Report 'Queue ID'" | awk '{print $1"  "$6"  "$7"  "$8"  "$9"
"$10"  "$11"  "$12"  "$13}'| less

To get the average queue delay in Second within an upadate interval k
less 123.txt | grep "Average Queue Delay Report 'Queue ID'" | awk '{print $1"  "$6"  "$7"  "$8"  "$9"
"$10"  "$11/1000"  "$12/1000"  "$13/1000}'| less

or

To calculate the average service class queue delay in sec (sc1avgdelay) // within an update time
interval k
less  123.txt | grep "QueueReport 'Queue ID'" | awk '{print $1"  "$8/1000"  "$9/1000"  "$10/1000}'|
grep -v 'nan'| less

============================================================
To get the average queue length, instant queue length and previous average queue length for a specific
service class queue within an upadate interval k

The first value represent the instant, the second represent the previous average and the third
represent the average queue length

less 123.txt | grep "SC1 Average Queue Length  'Queue ID'" | awk '{print $1"  "$6"  "$10"  "$11"
"$14"  "$15"  "$17"  "$19"  "$20"  "$22"  "$23}'| less

less 123.txt | grep "SC2 Average Queue Length  'Queue ID'" | awk '{print $1"  "$6"  "$10"  "$11"
"$14"  "$15"  "$17"  "$19"  "$20"  "$22"  "$23}'| less

less 123.txt | grep "SC3 Average Queue Length  'Queue ID'" | awk '{print $1"  "$6"  "$10"  "$11"
"$14"  "$15"  "$17"  "$19"  "$20"  "$22"  "$23}'| less


To get the average queue length within an upadate interval k For all classes in a specific queue
less 123.txt | grep "Average Queue Length Report 'Queue ID'" | awk '{print $1"  "$6"  "$7"  "$8"
"$9"  "$10"  "$11"  "$12"  "$13}'| less
============================================================
Number of dropped packet of SC1 at a queue
// Lower
less 123.txt | grep "'Queue ID'  SC1 Lthreshold Drop" | less | tail -1 | awk '{print $1"  "$6"  "$7"
"$8"  "$9"  "$10}' | less
// Higher
less 123.txt | grep "'Queue ID'  SC1 Hthreshold Drop" | less | tail -1 | awk '{print $1"  "$6"  "$7"
"$8"  "$9"  "$10}' | less
// overload
less 123.txt | grep "'Queue ID'  SC1 Buffer Drop" | less | tail -1 | awk '{print $1"  "$6"  "$7"
"$8"  "$9"  "$10}'| less
```

```
Number of dropped packet of SC2 at a queue
// Lower
less 123.txt | grep "'Queue ID'  SC2 Lthreshold Drop" | less | tail -1 | awk '{print $1"  "$6"  "$7"
"$8"  "$9"  "$10}' | less
// Higher
less 123.txt | grep "'Queue ID'  SC2 Hthreshold Drop" | less | tail -1 | awk '{print $1"  "$6"  "$7"
"$8"  "$9"  "$10}' | less
// overload
less 123.txt | grep "'Queue ID'  SC2 Buffer Drop" | less | tail -1 | awk '{print $1"  "$6"  "$7"
"$8"  "$9"  "$10}'| less

Number of dropped packet of SC3 at a queue
// Lower
less 123.txt | grep "'Queue ID'  SC3 Lthreshold Drop" | less | tail -1 | awk '{print $1"  "$6"  "$7"
"$8"  "$9"  "$10}' | less
// Higher
less 123.txt | grep "'Queue ID'  SC3 Hthreshold Drop" | less | tail -1 | awk '{print $1"  "$6"  "$7"
"$8"  "$9"  "$10}' | less
// overload
less 123.txt | grep "'Queue ID'  SC3 Buffer Drop" | less | tail -1 | awk '{print $1"  "$6"  "$7"
"$8"  "$9"  "$10}'| less
============================================================
// Total number of dropped packets for each drop case for whole simulation
less 123.txt | grep -i drop | awk '{print $7"  "$8"  "$9}'| sort | uniq -c
or
for ((i=1;i<4;i++)); do echo SC$i; less 123.txt | grep -i drop | grep SC$i | awk '{print $8" "$9}' |
sort | uniq -c; done

// Total number of dropped packets for each service class queue for whole simulation
less 123.txt | grep -i drop | awk '{print $7}'| sort | uniq -c
============================================================
// total number of dropped packet at a specific queue and for each case
less 123.txt | grep "'Queue ID'  "  | grep -i drop | awk '{print $7"  "$8"  "$9}'| sort | uniq -c

// total number of dropped packets at a specific queue for each service class
less 123.txt | grep "'Queue ID'  "  | grep -i drop | awk '{print $7}'| sort | uniq -c
============================================================
//Utilization
For a specific queue:
less 123.txt | grep "LinkUtilization in second for output port:  'Queue ID'" | awk '{print
$1/1000000000" "$2" "$3" "$4" "$5" "$6" "$7" "$8" "$9" "$10" "$11" "$12" "$13" "$14*100" "$15"
"$16*100" "$17" "$18*100" "$19" "$20*100}'| less
====================================================================

// For scenarios 2 and 3 only:

// To get the average queue delay of a sub-queue at queues (268,269 and 2610) or
(3810,3811,3812,3910,3911 and 3912)
 // for SC1
less 123.txt | grep "SC1GetAverageQueueDelay" | less          // in msec.
or
less 123.txt | grep "SC1UpdateAverageQueueDelay" | less       // in msec.
// for SC2
less 123.txt | grep "SC2GetAverageQueueDelay" | less
or
less 123.txt | grep "SC2UpdateAverageQueueDelay" | less
// for SC3
less 123.txt | grep "SC3GetAverageQueueDelay" | less
or
less 123.txt | grep "SC3UpdateAverageQueueDelay" | less

================================================
// To get the average queue length of sub queue at queues 268,269 and 2610 or
(3810,3811,3812,3910,3911 and 3912)
 // for SC1
less 123.txt | grep "SC1GetAverageQueueLength" | less
or
less 123.txt | grep "SC1UpdateAverageQueueLength" | less
// for SC2
less 123.txt | grep "SC2GetAverageQueueLength" | less
or
less 123.txt | grep "SC2UpdateAverageQueueLength" | less
// for SC3
less 123.txt | grep "SC3GetAverageQueueLength" | less
```

```
or
less 123.txt | grep "SC3UpdateAverageQueueLength" | less
==================================================
// To get the equivelent Minimum average sub queue delay  for all classes in queue 268 or 269 or 2610
or 3810 or 3811 or 3812 or 3910 or 3911 or 3912.
for each mentioned queue:
less 123.txt | grep "Average Delay Values 'X'" | less      //in msec.
X is either 268 or 269 or 2610 or 3810 or 3811 or 3812 or 3910 or 3911 or 3912.
==================================================
// To get the equivelent Maximum average sub queue length  for all classes in queue 268 or 269 or 2610
or 3810 or 3811 or 3812 or 3910 or 3911 or 3912.
for each mentioned queue:
less 123.txt | grep "Average Length Values 'X'" | less
X is either 268 or 269 or 2610 or 3810 or 3811 or 3812 or 3910 or 3911 or 3912.
==================================================
// To get the average service class queue delay at a specific queue (268 or 269 or 2610 or 3810 or
3811 or 3812 or 3910 or 3911 or 3912) and the equivelent Minimum average delay value.
less 123.txt | grep "delay'X':" | less            // in msec.
X is either 268 or 269 or 2610 or 3810 or 3811 or 3812 or 3910 or 3911 or 3912.
==================================================
// To get the average service class queue length at a specific queue (268 or 269 or 2610 or 3810 or
3811 or 3812 or 3910 or 3911 or 3912) and the equivelent Maximum average length value.
less 123.txt | grep "length'X':" | less
==================================================
// To get the equivelent average of average scheduling rates for service class queues at queues
(268,269 and 2610) or (3810,3811,3812,3910,3911 and 3912)
less 123.txt | grep "CalculateSchedulingRateByAverage" | less
==================================================
// To get the equivelent Minimum average delay queue for service class queues at queues 268 or 269 or
2610 or (3810,3811,3812,3910,3911 and 3912)
less 123.txt | grep "Average Delay Values " | less        // in msec.
==================================================
// To get the equivelent Maximum average length queue for service class queues at queues 268 or 269 or
2610 or (3810,3811,3812,3910,3911 and 3912)
less 123.txt | grep "Average Length Values " | less
==================================================
```

A-9: Conferences and Training Courses.

No.	preliminary study and Training courses	Date
1	Introduction to Endnote X7	9th of December 2013
2	Qualitative research tools: Nvivo. Day 1 introduction to Nvivo.	11th of February 2014
3	Doing a literature review.	19th of February 2014
4	Postgraduate students: Enterprise futures conference.	29th of May 2014
5	Attending NTAD MSC module (Network, technology, design and architecture).	1st semester 2013/2014
6	Attending MSc Module NPS (Network programming and simulation).	2nd semester 2013/2014
7	Attending PGR Conference 2014-Salford University.	22nd-23rd of January 2014
8	Submitted a poster and abstract in the College Dean's Annual Research Showcase Event.	18th of June 2014
9	Attending symposium (PGNET2014, Liverpool).	23rd-24th of June 2014
10	LEAP Higher - Academic Writing in English as a Second Language.	24th of Nov, 1st of Dec, 8th of Dec. and 15th of Dec. 2014
11	How to get Published with IEEE.	17th of Feb 2015.
12	IP EXPO 2015 – Manchester.	20th and 21st of May 2015.
13	Submitted a poster in SPARC/Dean's Annual Research Showcase – Salford University.	26th to 28th of May 2015.
14	Attended GTS workshops (Inclusive teaching & classroom management) Assessment and Feedback, Salford University.	13th of January 2017
15	Submitting a paper to IFIP Networking 2017 conference sponsored by IEEE Computer Society, Stockholm – Sweden.	12th-16th June 2017
16	Participating in the University of Salford conference (SPARC 2017)	27th -29th June 2017